Sistemas Operativos

Modelo de sistema Cliente/Servidor - Parte 2

Juan Carlos Fernández Caballero
jfcaballero@uco.es

Asignatura "Sistemas Operativos"
2º de Grado en Ingeniería en Informática
Dpto. Informática y Análisis Numérico
Universidad de Córdoba

24 de noviembre de 2016

Objetivos

- Comprender **cómo funciona el modelo cliente/servidor** y cómo se **organiza la comunicación.**
- Conocer y comprender **las alternativas y variantes del modelo cliente/servidor**.
- Analizar el modelo cliente/servidor, discutiendo todos los **elementos que intervienen en la comunicación** y las diferentes **decisiones de diseño** que implica.

Índice

Modelos y tecnologías de comunicación más usuales

Modelos:

- Cliente-Servidor (C-S) → Lo usan muchas tecnologías de comunicación y es la base de otros modelos.
- Editor/subscriptor → Muy utilizado en "Internet de las cosas".
- Comunicación en grupo (Punto a punto o *P2P*, *Broadcast*).
- Maestro-Esclavo.

Tecnologías:

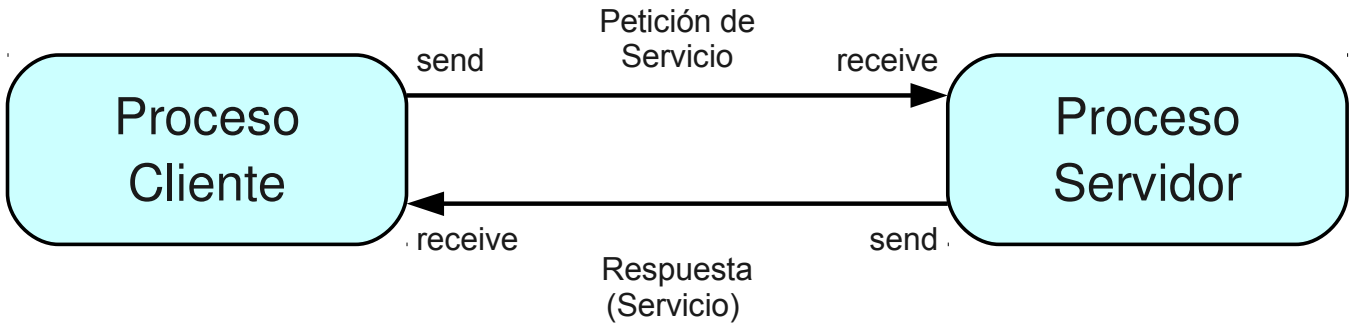
- Paso de mensajes: MPI, PVM. Este último en desuso.
- Llamadas a procedimientos remotos (RPC): RPC de Sun, DCE, RMI de Java.
- Servicios Web: SOAP, Rest → Basado en RPC.
- Objetos distribuidos: CORBA.

Modelo cliente-servidor

- Modelo en el que **las tareas se reparten** entre los **proveedores** de servicios - servidores -, y los **demandantes** - clientes -.
- Se basa en un modelo solicitud-respuesta. El cliente **pide servicio** y el servidor **hace el trabajo** y **regresa datos** o un **código de error**.
- Una máquina puede ejecutar uno o **varios procesos clientes**, uno o **varios procesos servidores**.
- Los **servidores** pueden ser también **clientes de otros servicios**. Ejemplo: **Servidores Web** como **clientes del servicio DNS** para la traducción de nombres de dominio.

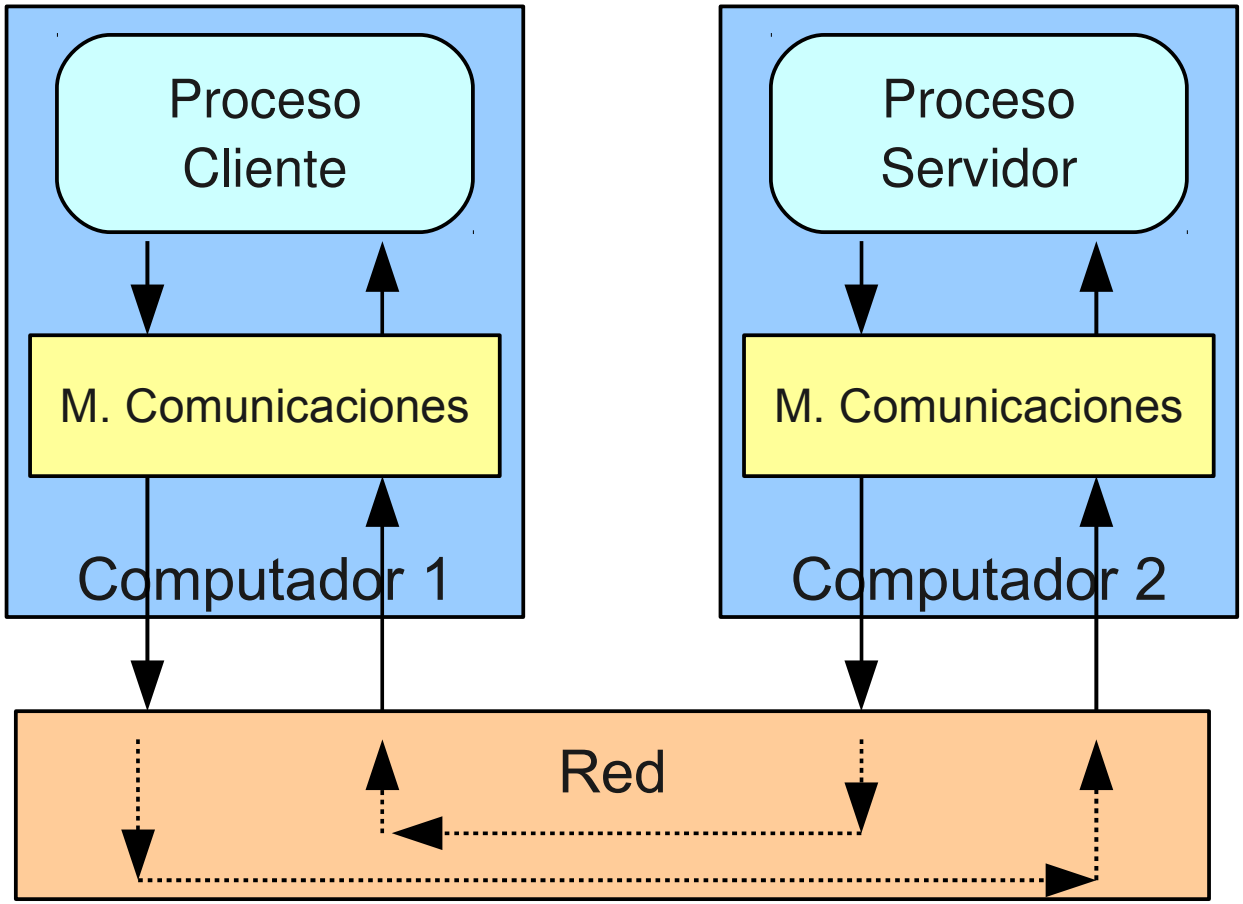
Modelo cliente-servidor

- Se establece conexión solo **cuando se utilice**.
- Necesidad de **dos primitivas** para solicitud-respuesta de un servicio:
  - Una para **enviar** mensajes - send().
  - Una para **recibir** mensajes - receive().

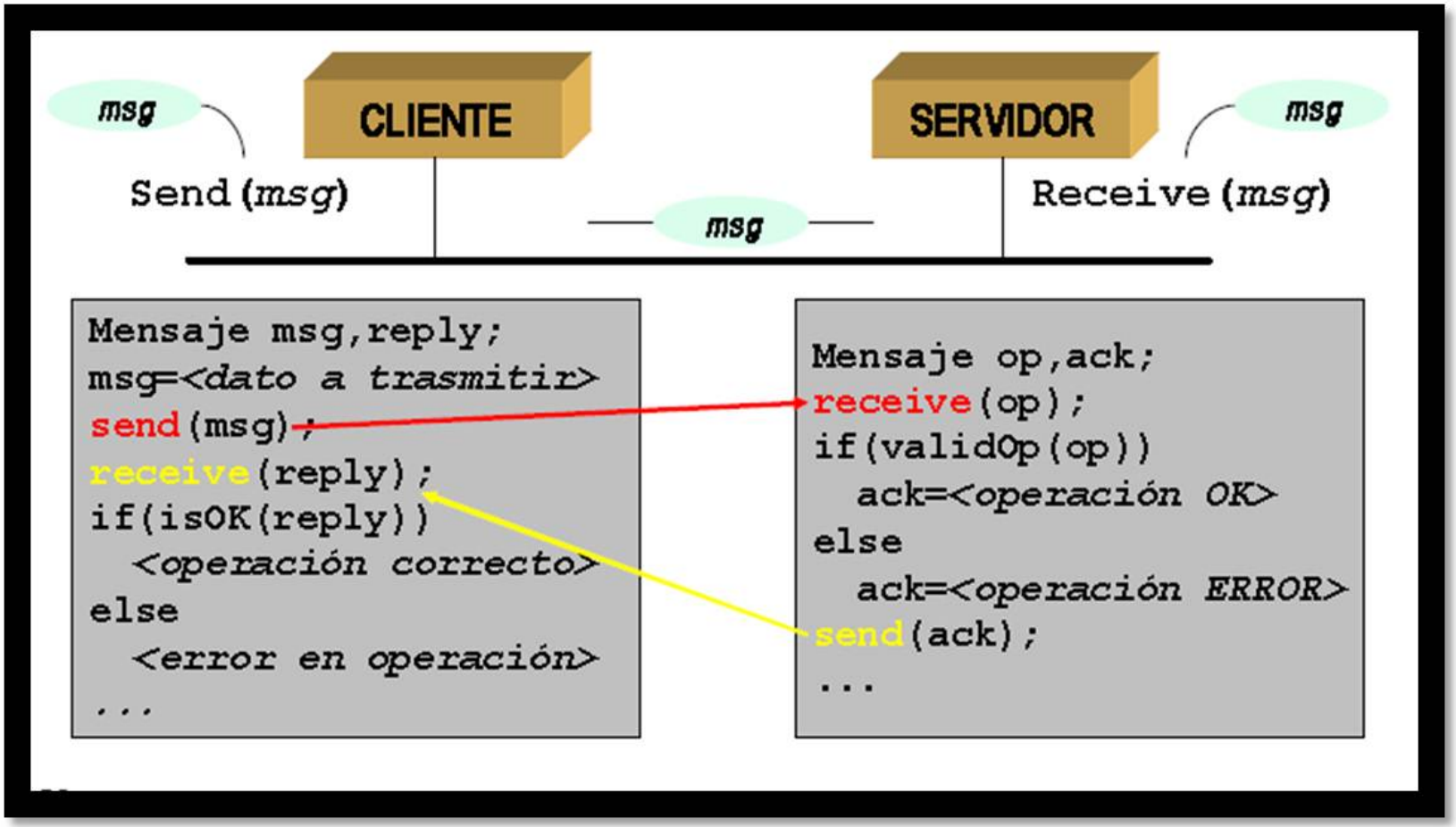


- Comunicación entre extremos a través del **módulo de comunicaciones del núcleo** del sistema operativo → Transparencia por medio de API.

Modelo cliente-servidor



Modelo cliente-servidor

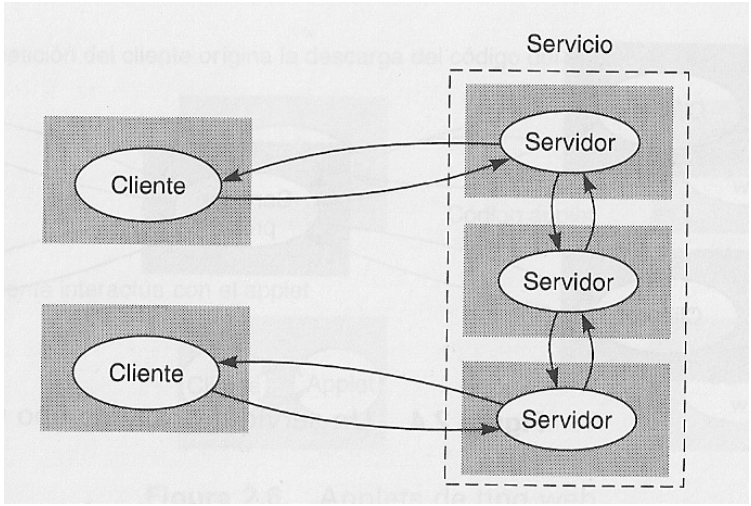


Mejora de prestaciones en modelo Cliente-Servidor

¿Cómo mejorar la disponibilidad y tolerancia a fallos?

1. **Dividir el conjunto de objetos** en los que está basado el servicio en varias máquinas.
2. **Mantener copias replicadas** en varias máquinas.

   Ejemplos: Servidores Web replicados, DNS.



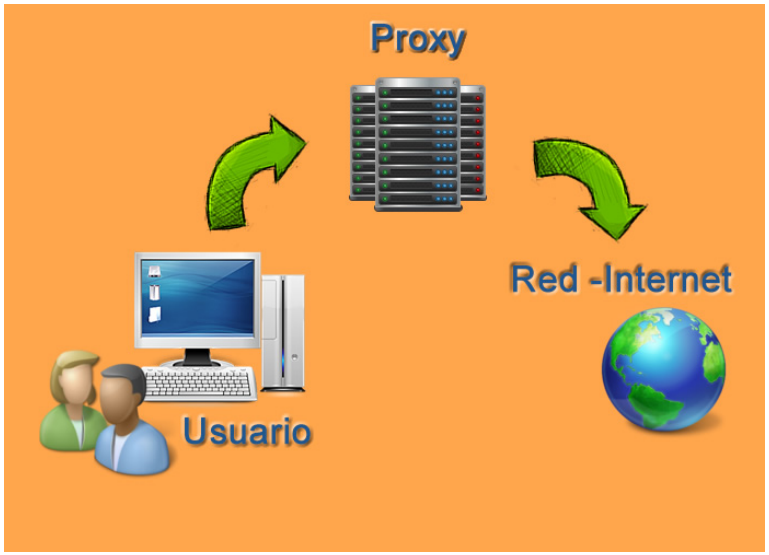
Mejora de prestaciones en modelo Cliente-Servidor

3. Uso de **cachés o servidores proxy**.
   - Programa o dispositivo **intermediario** entre el cliente y el servidor. Lo pueden usar tanto clientes como servidores.
   - Dos tipos genéricos: Proxy local y proxy de red o externo.
   - Consultar `https://es.wikipedia.org/wiki/Proxy`
   - Funcionalidades: **Restricción de tráfico actuando como cortafuegos** (no deja enviar/recibir peticiones prohibidas), **mejora de respuesta** (caché Web, caché de servicios en servidor), **anonimato de la comunicación** (no se sabe de quién proviene una petición), **distribución de carga** (envío de petición a un servidor u otro dependiendo de su carga), etc.

Mejora de prestaciones en modelo Cliente-Servidor

Proxy local

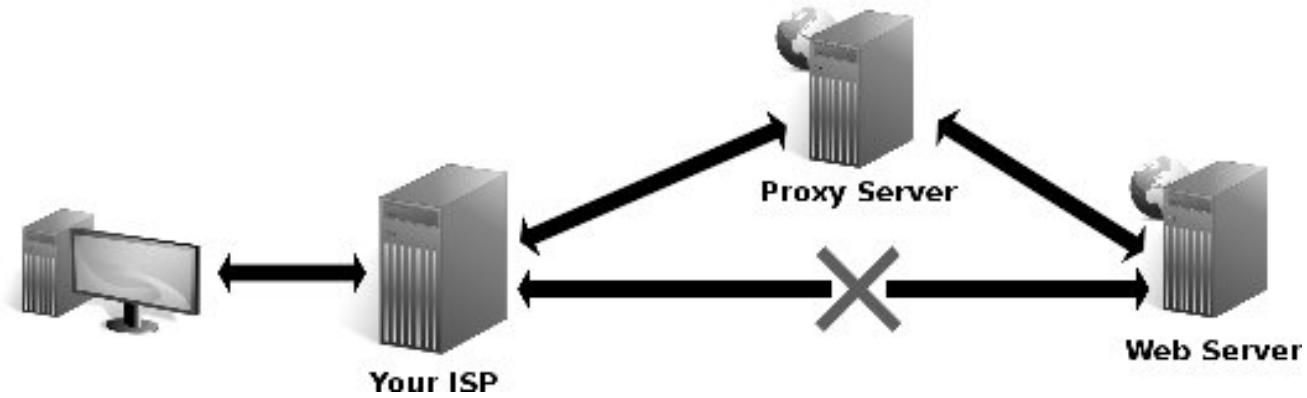
- Software ubicado en la misma máquina que peticiona o sirve.
- A veces nombrado como *proxy* o servidor **NO dedicado**.
- En la **figura** el *proxy* intermedio sería **la propia maquina del usuario**.



Mejora de prestaciones en modelo Cliente-Servidor

Proxy de red

- **Máquina intermedia** entre cliente y servidor, nombrado a veces como *proxy* **dedicado**.
- **Como caché:** Cuando un cliente recibe un **servicio**, este **se añade a la caché del proxy** $\rightarrow$ Incremento de la disponibilidad de los servicios, reduciendo la carga de la red y del servidor.



Código móvil (No computación móvil)

Código móvil: Aquel código o programa que **se traslada de un servidor a un cliente** y que se **ejecuta interactuando en el cliente**: Applets de Java, animaciones en Flash...

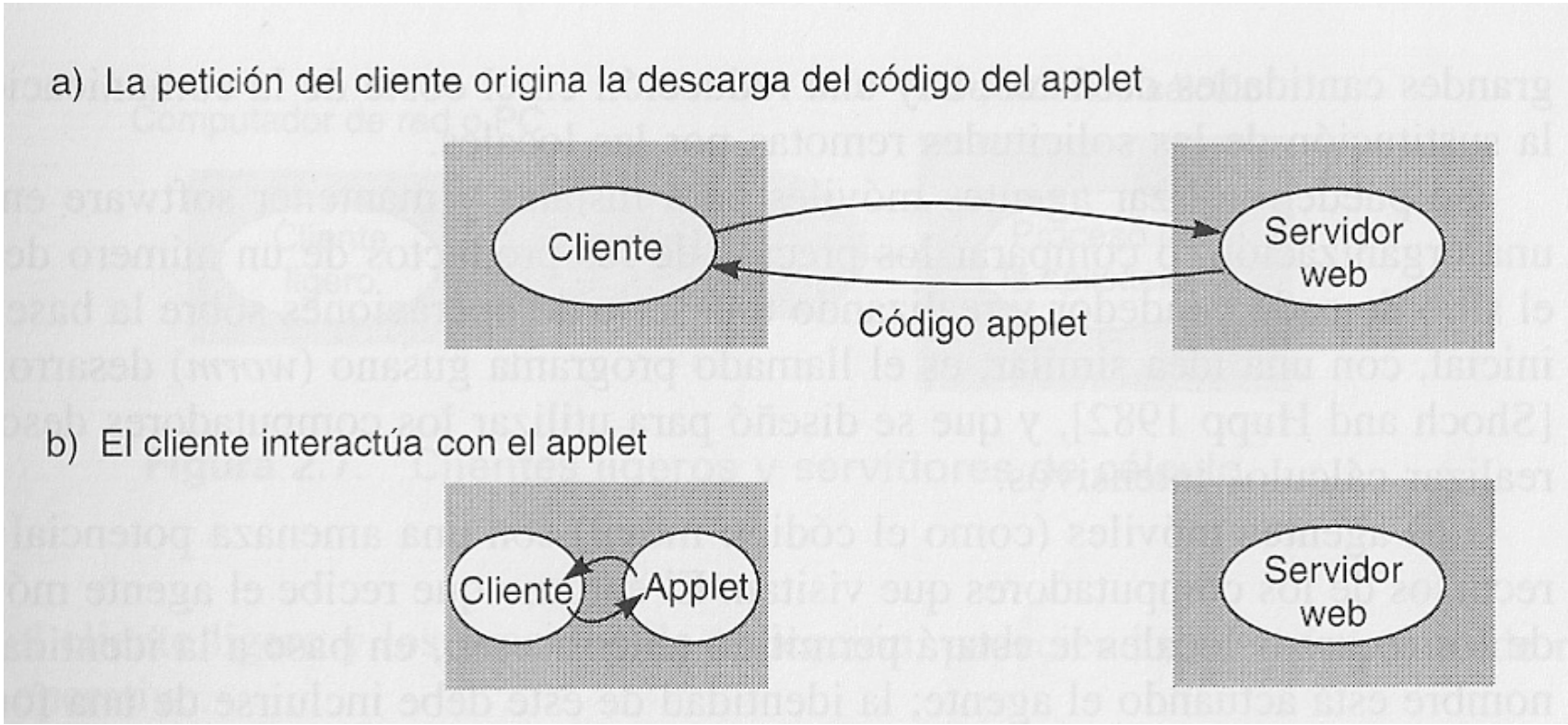
Ventajas:

- Mejora en **servicios** que requieren una **respuesta interactiva**.
- **No se sufren retardos**, variabilidad en el ancho de banda ni problemas de red.

Desventajas:

- Puede haber **problemas de seguridad**: Código malicioso ejecutado en el cliente.

Código móvil (No computación móvil)



Agentes móviles

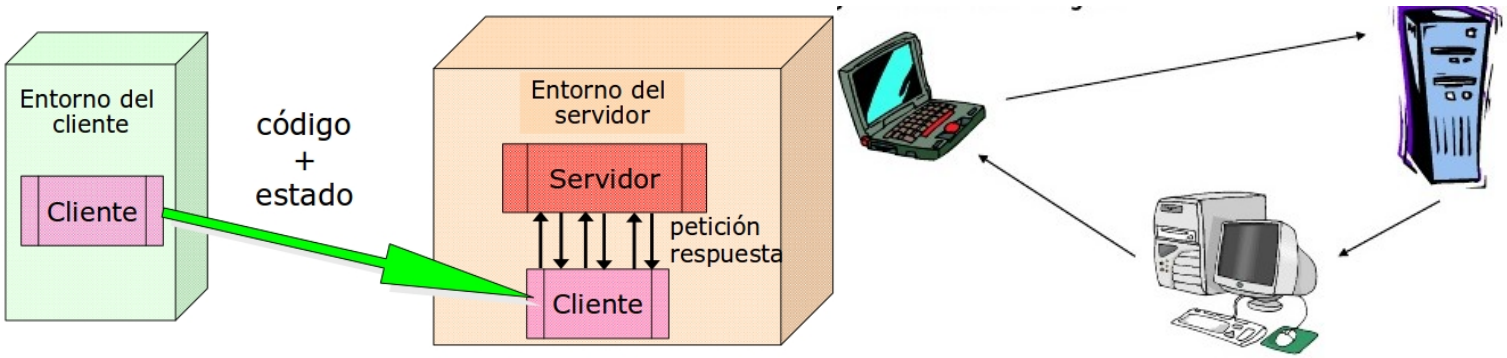
Agente móvil: Programa en ejecución **(Código + Datos)** y en movimiento por la red.

- **Se traslada** de un computador a otro de la **red** haciendo tareas para su **propietario cliente**.
- Las tareas se hacen mediante **solicitudes a los recursos locales en otras máquinas**.

  Ejemplos: Descubrir la configuración de una red, monitorización y detección de fallos en los nodos de una red, instalación y mantenimiento software de una organización, etc.
- Necesitan **acreditación previa** de la máquina que visita.
- **Conversión del código del agente** en la máquina que visita y **posterior ejecución** local.

Agentes móviles

- **Comunicación asíncrona**: El agente vuelve cuando termine.
- Posible amenaza de seguridad: Código malicioso, suplantaciones de identidad, cambios de código que alteren su funcionalidad.



Computadores de red

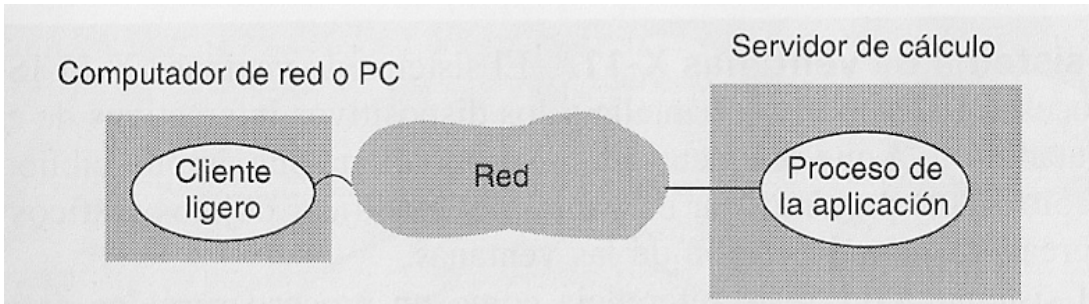
Computador de red: Aquel que **descarga su sistema operativo** y **cualquier programa de aplicación** software que necesite el usuario **desde un servidor remoto** cada vez que **arrancamos el equipo**.

- Se mantiene una caché local (disco duro) con los archivos remotos cargados recientemente (mejora el rendimiento).
- Permite al usuario **migrar de un equipo físico a otro.**
- Mejora la **gestión y mantenimiento del software** por parte del **administrador**.

Clientes ligeros

Cliente ligero: Es una **capa software** que **ejecuta** únicamente **una interfaz gráfica en un cliente** que **interactúa con un servidor**.

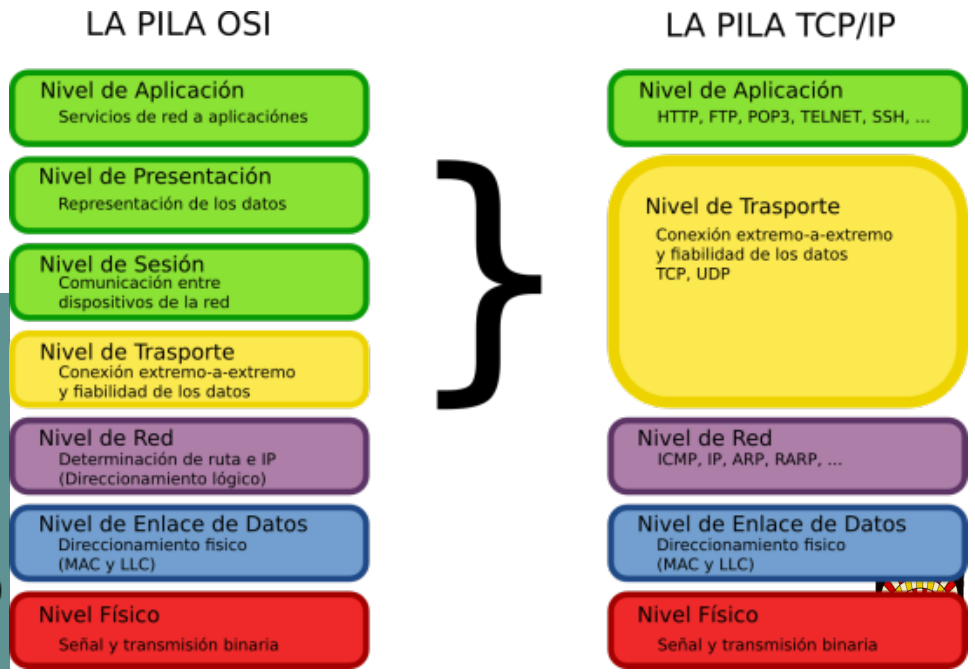
- Los **programas** de aplicación **ejecutan en un servidor remoto o cluster** mediante modelo cliente-servidor.
- **Problemas de latencia en aplicaciones gráficas** muy interactivas.
- Mejora la **gestión y mantenimiento del software** por parte del **administrador**.
- Ejemplos: **Cliente ICA para Windows UCO**, control de computadores remotos con VNC (escritorio remoto), CAD.



Protocolo de comunicación

Protocolo → Acuerdo entre las partes en cómo debe desarrollarse la comunicación. Se utiliza un **subconjunto protocolos** de la pila ISO/OSI y soporta TCP y UDP.

- 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)



Identificación del servidor

¿**Cómo identifica el cliente el proceso servidor de un servicio**?
Dos esquemas básicos:

1. ID máquina + ID number process.
   - Se usa **ID Máquina (IP)** para encontrar la máquina.
   - Se usa **ID number** para encontrar al proceso que ofrece el servicio en esa máquina.
   - **Falta transparencia** para el usuario, *ID Máquina (IP)* + *ID number* se deben indicar **explícitamente** por parte del cliente:
     - En el código a compilar (recompilación en cambios).
     - En ficheros de configuración o linea de argumentos.
   - **Débil ante cambios:** Si la máquina servidora y/o los procesos se cambian hay que cambiar el cliente.

Identificación del servidor

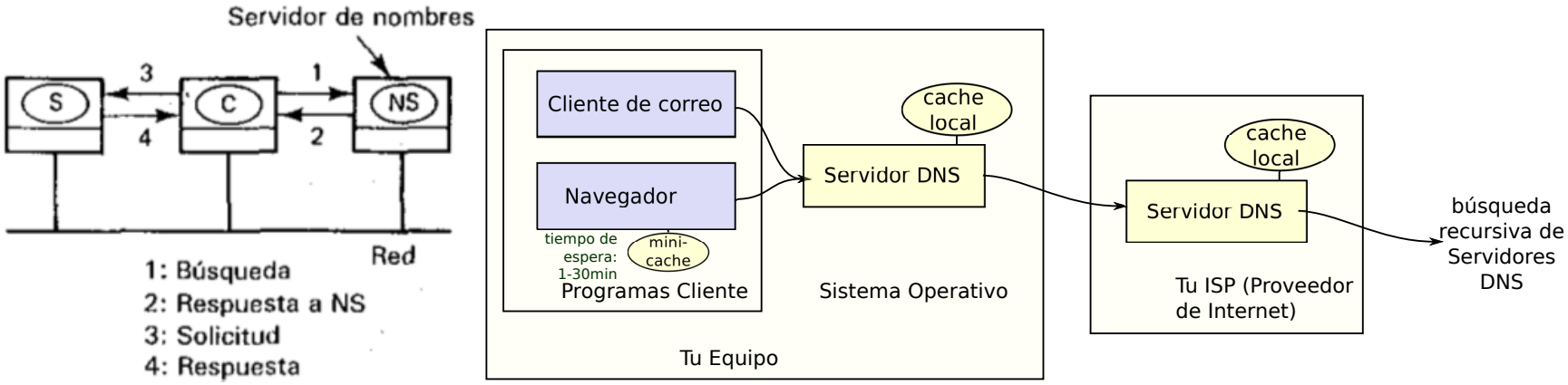
2. Servidor de Nombres de Servicio:

   Servidor especial encargado de **asociar un nombre de servicio** (ASCII) a cada una de las máquinas que lo ofrecen.
   ID Máquina (IP) + *ID Servicio* = cadena ASCII.

   - En el primer intento el **cliente envía una solicitud** a un **servidor especial** de asociaciones.
   - El **servidor le devuelve la dirección** de envío.
   - **Solicitud** y **respuesta** Cliente-Servidor.
   - **Mayor transparencia** y flexible a cambios.
   - Problema: Si se cae el **servidor especial centralizado** se cae el sistema → Necesidad de **replicas** al ser centralizada.

Identificación del servidor

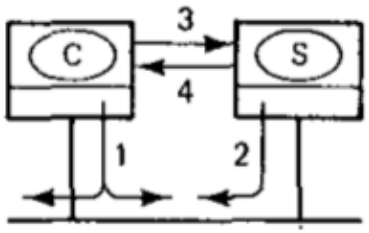
`http://es.wikipedia.org/wiki/Domain_Name_System`



Identificación del servidor

Direccionamiento en LAN → Broadcast a la red.

- Envío de **paquete de localización de servicio** en la red → *Broadcast*.
- **Recepción** del paquete por parte de **todas las maquinas** para **identificar el servicio**.
- **Devolución de su dirección en la red** de la máquina servidora que tiene el servicio.
- **Solicitud** y **respuesta**.

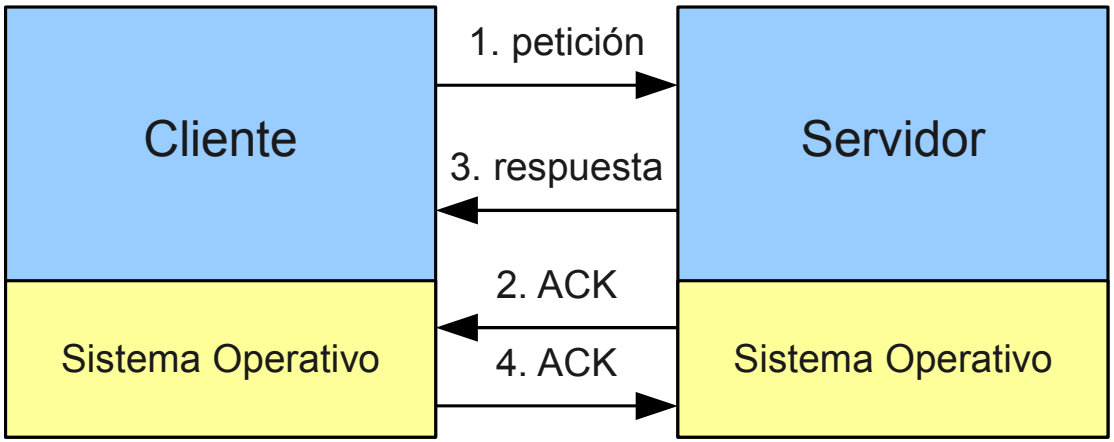


1: Transmisión
2: Aquí estoy
3: Solicitud

Reconocimiento de mensajes

Inciso: Antes de pasar a estudiar las primitivas con/sin bloqueo, este sería el **comportamiento básico e idóneo** en una **interacción Cliente-Servidor.**

Mensajes reconocidos de forma individual:

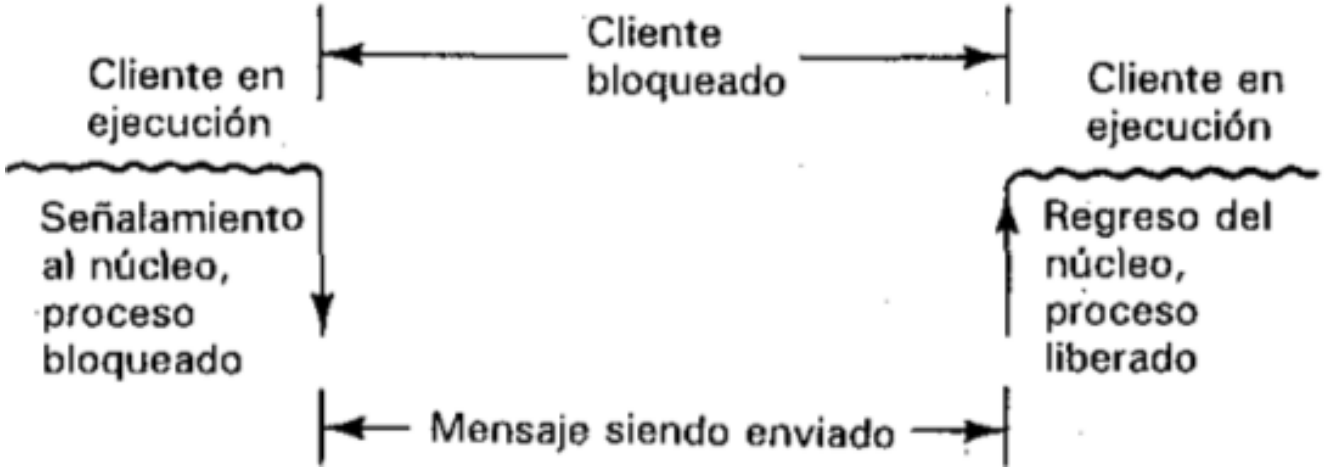


Primitivas síncronas CON bloqueo

- Una llamada a *send(A,mensaje) bloqueante* deja en **estado bloqueado al emisor** → NO puede ejecutar la siguiente instrucción **hasta que el receptor** no devuelva el **ACK**.
  - *send(A,mensaje)*. El núcleo del S.O junto con la API se encarga de averiguar si se ha recibido el mensaje ACK, para pasar el emisor de Bloqueado a estado Listo.
- Una llamada a *receive(B,mensaje) bloqueante* deja en **estado bloqueado al receptor** → NO puede ejecutar la siguiente instrucción hasta que no llegue un **mensaje de petición y se envíe ACK** o un **mensaje de respuesta y se envíe ACK** (depende de si es cliente o servidor).
  - *receive(B,mensaje)*. El núcleo del S.O junto con la API se encarga de averiguar si se ha recibido una petición o una respuesta, mandar el ACK y para pasar de estado Bloqueado a estado Listo.

Primitivas síncronas CON bloqueo

- Ejemplo de petición *send(A,mensaje)* desde cliente a servidor y bloqueo.
- Recepción en núcleo cliente del ACK del servidor y paso a Listo.



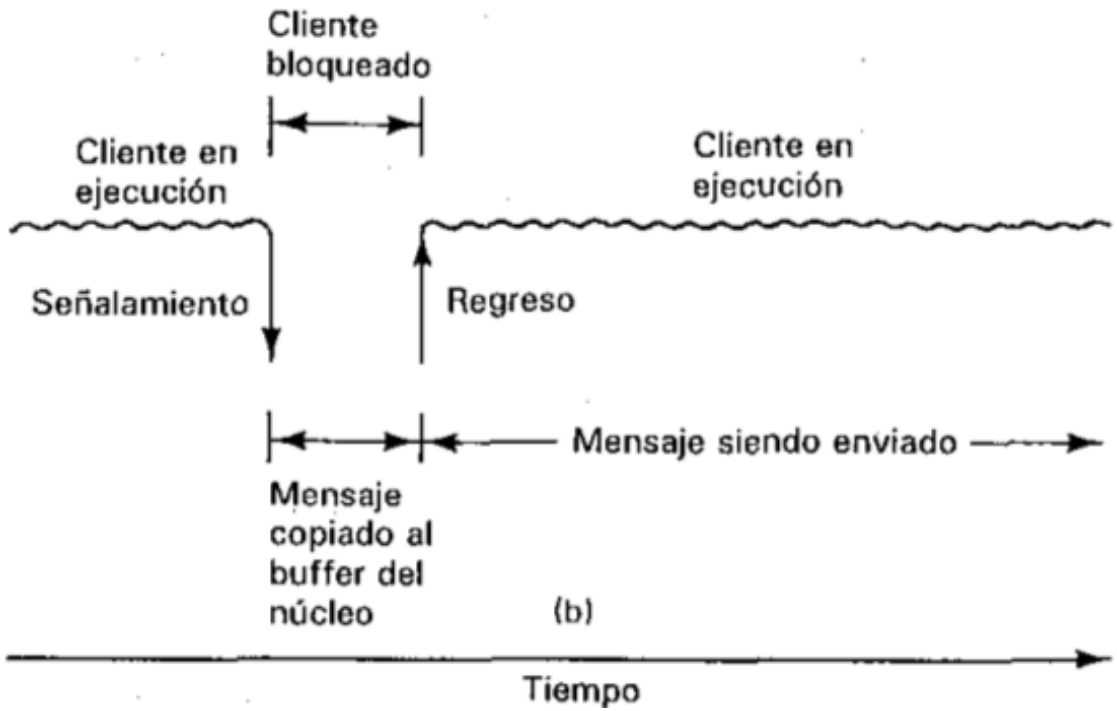
Primitivas asíncronas SIN bloqueo

Lado del EMISOR:

- El emisor **puede continuar con otras instrucciones** mientras el mensaje se envía con *send(A,mensaje)* y vuelve el ACK.
  - Problema $\rightarrow$ **No se debe modificar el mensaje** (*buffer* que lo contiene) hasta que el receptor lo reciba.
- Solucion1: **Copia a buffer interno** del núcleo. Esto lleva tiempo de cómputo y duplicidad de información.
- Solución2: **Interrumpir al proceso** cuando se haya recibido el (ACK), indicando así que ya puede usar el buffer que contiene el mensaje.
  - Problema a Sol.2 $\rightarrow$ **Comprobación de interrupciones** en cada ciclo, además la **programación es difícil** y poco reproducible para encontrar fallos en depuración.

Primitivas asíncronas SIN bloqueo

- Ejemplo de petición *send(A,mensaje)* desde cliente a servidor y continuación.

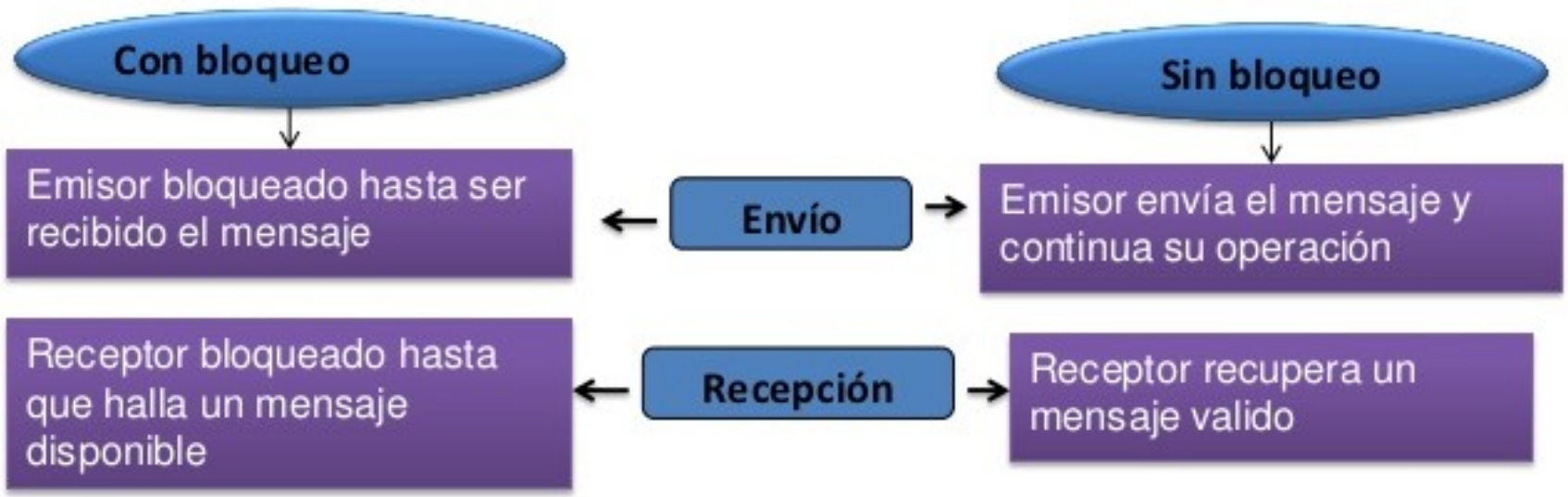


Primitivas asíncronas SIN bloqueo

Lado del RECEPTOR:

- El **receptor puede ejecutar otras instrucciones** después de la llamada a *receive(B,mensaje)*, y no esperar un **mensaje de respuesta** o el **mensaje ACK** del emisor.
- Un *receive(B,mensaje)* sin bloqueo **le indica al núcleo del S.O la localización del buffer donde poner el mensaje** cuando llegue.
  - Problema: $\rightarrow$ **Comprobación de interrupciones** en cada ciclo o **primitivas de testeo** del núcleo que indiquen que ya se ha recibido el mensaje de respuesta o de ACK del emisor.

Resumen gráfico



Primitivas almacenadas/no almacenadas

Independientemente de si las primitivas son síncronas o asíncronas, imaginemos lo siguiente:

La **llamada** *send(A,mensaje)* del **cliente** se produce **antes** que la **llamada** *receive(B,mensaje)* del **servidor** (p.ej. cuando el servidor está atendiendo a otros clientes o haciendo cierto cálculo).

¿Qué hace el servidor con el mensaje que le llega?

- Descartarlo → **Primitiva no almacenada**.
- Guardarlo → **Primitiva almacenada**:
  1. Guardarlo con un **tiempo de expiración - timeout**.
  2. Utilizar buffer-buzón de recepción → ¿y si el buzón se llena?.
     - Mandar un mensaje al cliente → Inténtalo de nuevo más tarde.

Primitivas fiables y no fiables

- Primitivas fiables: Aquellas en las que cuando hacemos un *send(A,mensaje)* **QUIEREN ASEGURAR que el mensaje ha llegado al receptor.**
- Primitivas no fiables: Aquellas en las que cuando hacemos un *send(A,mensaje)* **NO GARANTIZAN que el mensaje haya llegado al receptor**.

¿Cómo se pueden conseguir primitivas fiables?

- Introduciendo **mensajes de comportamientos inesperados** en los **protocolos a seguir** por emisor y receptor $\rightarrow$ Se verá a continuación.
- **Transparente al programador**, los núcleos interactúan con la API para llevar a cabo el protocolo e intercambio de mensajes.

Primitivas fiables

Problemas en primitivas fiables

Si no llega un ACK del SERVIDOR AL CLIENTE, puede deberse:

- Petición perdida.
- Petición recibida, pero la respuesta se pierde.
- Retraso de la respuesta por cálculos en servidor.

Solución:

- Se pone un **timeout en el S.O. del cliente**, si la respuesta del proceso servidor no llega a tiempo, la petición se retransmite.
- Si el servidor recibe **dos peticiones descarta la última** (debido a retraso por cálculos).

Primitivas fiables

Problemas en primitivas fiables

Si no llega un ACK del CLIENTE AL SERVIDOR, puede deberse:

- Respuesta perdida.
- ACK del cliente perdido.

Solución:

- Se pone un **timeout en el S.O. del servidor**. El servidor debe **guardar** una **copia de la respuesta**, por si se pierde durante el envío y **debe reenviarla**.
- Si el cliente recibe **dos respuestas se descarta la última**.

Tipos de mensajes

Algunos tipos de mensajes utilizados en el modelo Cliente-Servidor en **interacciones NO idóneas**. Son transparentes al programador. El núcleo de S.O se encarga de ello.

Código	Tipo de paquete	De	A	Descripción
REQ	Solicitud	Cliente	Servidor	El cliente desea servicio
REP	Respuesta	Servidor	Cliente	Respuesta del servidor al cliente
ACK	Reconocimiento	Cual-quiera	Algún otro	El paquete anterior que ha llegado
AYA	¿Estás vivo?	Cliente	Servidor	Verifica si el servidor se ha descompuesto
IAA	Estoy vivo	Servidor	Cliente	El servidor no se ha descompuesto
TA	Intenta de nuevo	Servidor	Cliente	El servidor no tiene espacio
AU	Dirección desconocida	Servidor	Cliente	Ningún proceso utiliza esta dirección

Tipos de mensajes

- Lo más frecuente es que el **AYA se envíe del cliente al servidor**.
- **TA** puede provocarse, p.ej., por estar el **buzón de recepción del servidor lleno**.
- El mensaje **AU**, además del servidor, también lo puede envíar el S.O. del **cliente hacia el servidor** cuando la **respuesta no es para ese cliente**.
- Incluso **AU**, también lo puede enviar el S.O. del **cliente hacia si mismo** cuando **no se encuentra la dirección de un servidor**.

División en paquetes

- **Los mensajes son divididos en distintos paquetes** que se envían por separado.
- **Estrategias de reconocimientos ACK** de los paquetes:
  - Un ACK por cada paquete.
    ☹ Generamos más tráfico de mensajes.
    ☺ Sabemos el paquete exacto que falló y sólo se nos debe reenviar ese paquete.
  - Un ACK al final del mensaje.
    ☺ Menor tráfico de mensajes.
    ☹ Si algo falla, hay que reenviar todo el mensaje.

Formato de los datos

- Muchos fabricantes no se ponen de acuerdo en cómo **representar los datos**, big endian, little endian...
- Soluciones:
  1. Utilizar un estándar de representación de datos que todos usen en el envío y recepción.
     ☹ Se puede llegar al caso extremo de realizar **conversiones no necesarias** (equipos homogéneos).
  2. Todos los procesos **conocen todos los sistemas de representación** y hacen las conversiones.
     ☹ Necesario implementar una **gran cantidad de métodos de conversión**.

Bibliografía

Andrew S. Tanembaum.
Sistemas Operativos Distribuidos (Primera Edición).
Tema 2. Comunicación en los Sistemas Distribuidos.
Sección 2.3. El modelo Cliente-Servidor.
Prentice Hall, 1996.

George Colouris, Jean Dollimore y Tim Kindberg.
Sistemas Distribuidos. Conceptos y Diseño (Tercera Edición).
Tema 2. Modelos de Sistema.
Addison Wesley, 2001.

Sistemas Operativos

Modelo de sistema Cliente/Servidor - Parte 2

Juan Carlos Fernández Caballero
jfcaballero@uco.es

Asignatura "Sistemas Operativos"
2º de Grado en Ingeniería en Informática
Dpto. Informática y Análisis Numérico
Universidad de Córdoba

24 de noviembre de 2016