# SCD.

## **Tarea 4:** Modelo cliente-servidor

**Padilla Perez Jorge Daray**

**02/09/2024**

**CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS**

**Prof. Gutiérrez Salmerón Martha del Carmen**

# Contenido

# Introducción

La comunicación cliente-servidor es un concepto fundamental en el ámbito de las redes y sistemas distribuidos. Se refiere a la interacción entre dos entidades: el cliente (que solicita servicios o recursos) y el servidor (que proporciona esos servicios o recursos). A través de protocolos y primitivas, esta comunicación permite que aplicaciones, sitios web y servicios se conecten y compartan información de manera eficiente.

# Contenido

## Modelo cliente-servidor

Este modelo ya no es por capas, sugiere estructurar al sistema de operación como un grupo de procesos cooperantes: Clientes y Servidores. Escribe la definición y características de cada uno.

**Clientes**:

- **Definición**: Los clientes son componentes de un sistema que solicitan y utilizan recursos o servicios proporcionados por otros componentes. En el contexto de redes y sistemas distribuidos, los clientes son programas o dispositivos que se comunican con servidores para obtener información o realizar acciones.
- **Características**:
    - **Pasivos**: Los clientes generalmente son pasivos en el sentido de que esperan respuestas o servicios de los servidores.
    - **Iniciadores de solicitudes**: Los clientes envían solicitudes a los servidores para obtener datos o realizar operaciones específicas.
    - **Interfaz de usuario**: Los clientes suelen tener una interfaz de usuario que permite a los usuarios interactuar con el sistema.
    - **Ejemplos**: Navegadores web (como el que estás usando ahora), aplicaciones de correo electrónico, aplicaciones de mensajería, etc.

**Servidores**:

- **Definición**: Los servidores son componentes que proporcionan servicios o recursos a los clientes. Están diseñados para escuchar y responder a las solicitudes de los clientes.
- **Características**:
    - **Activos**: Los servidores son activos y están siempre disponibles para procesar solicitudes entrantes.
    - **Recursos compartidos**: Proporcionan recursos (como archivos, bases de datos, servicios web, etc.) a los clientes.
    - **Escalabilidad**: Los servidores pueden escalar para manejar múltiples conexiones simultáneas.
    - **Ejemplos**: Servidores web (como los que alojan sitios web), servidores de correo, servidores de bases de datos, etc.

  Para evitar el costo excesivo de los protocolos orientados a la conexión, usualmente utiliza un protocolo “requerimiento- respuesta” sin conexión.

## Modelo “Requerimiento-Respuesta” sin Conexión:

- En este modelo, las comunicaciones entre los componentes (clientes y servidores) no requieren una conexión persistente. Cada solicitud se trata de manera independiente, sin mantener un estado entre ellas.
- Cuando un cliente necesita algo (por ejemplo, datos o servicios), envía una solicitud al servidor. El servidor procesa la solicitud y envía una respuesta al cliente.
- No se establece una conexión prolongada; en cambio, se crea una conexión temporal solo para la transacción específica.
- Ejemplo: El protocolo HTTP (Hypertext Transfer Protocol) utilizado en la web sigue este enfoque. Cada solicitud HTTP (como cargar una página web o descargar un archivo) es independiente y no mantiene una conexión persistente.

2. **Ventajas del Modelo “Requerimiento-Respuesta” sin Conexión**:
   - **Sencillez**: Al no requerir una conexión persistente, la implementación es más simple. No hay necesidad de establecer, mantener y cerrar conexiones complejas.
   - **Eficiencia**: No se incurre en el costo adicional de mantener conexiones abiertas. Esto es especialmente útil en redes con ancho de banda limitado o alta latencia.
   - **Pila de Protocolo Más Corta**: Dado que no hay necesidad de negociar y mantener conexiones, la pila de protocolo es más corta y más eficiente.
   - **Independencia de Estado**: Cada solicitud es independiente, lo que facilita la escalabilidad y la distribución de carga.
   - **Menos Overhead**: No se necesita el seguimiento del estado entre solicitudes, lo que reduce el overhead.
3. **Niveles de Protocolos**:
   - Si todas las máquinas fueran idénticas y solo se necesitaran tres niveles de protocolos, podríamos simplificar la comunicación:
     1. **Nivel de Aplicación**: Donde se definen las reglas específicas para las aplicaciones (por ejemplo, HTTP, SMTP, FTP).
     2. **Nivel de Transporte**: Para garantizar la entrega confiable de datos (por ejemplo, TCP o UDP).
     3. **Nivel de Red**: Para enrutar paquetes entre redes (por ejemplo, IP).

## Modelo OSI vs Modelo cliente-servidor

Debido a esta estructura sencilla, se pueden reducir los servicios de comunicación que presta el micronúcleo reduce a dos llamadas al sistema, una para el envío de mensajes y otra para la recepción.

1. Estas llamadas al sistema se pueden pedir a través de procedimientos de biblioteca, como **send(dest,&mptr) y receive(addr, &mptr).**

   **send(dest, &mptr):**

   - **Definición**: La llamada al sistema send se utiliza para enviar un mensaje desde un proceso (cliente) a otro proceso (servidor) dentro del sistema operativo.
   - **Características**:
     - Destino (dest): Especifica el identificador o dirección del proceso receptor al que se enviará el mensaje.
     - Mensaje (&mptr): Contiene los datos que se desean enviar al proceso destino.
     - Bloqueante o No Bloqueante: Dependiendo de la implementación, la llamada send puede ser bloqueante (espera hasta que el mensaje se entregue) o no bloqueante (retorna inmediatamente).
     - Comunicación Síncrona o Asíncrona: Puede ser parte de una comunicación síncrona (espera respuesta) o asíncrona (no espera respuesta).
     - Ejemplo: En sistemas de mensajería, como sistemas de colas, send se utiliza para poner mensajes en la cola para su posterior procesamiento.

2. **receive(addr, &mptr):**
   - **Definición**: La llamada al sistema receive se utiliza para recibir mensajes entrantes en un proceso.
   - **Características**:
     - Dirección (addr): Indica desde qué proceso se espera recibir el mensaje. Puede ser un identificador o un filtro.
     - Mensaje (&mptr): Almacena el contenido del mensaje recibido.
     - Bloqueante o No Bloqueante: Al igual que send, receive puede ser bloqueante o no bloqueante.
     - Comunicación Síncrona o Asíncrona: Puede esperar una respuesta síncrona o simplemente recibir mensajes sin esperar.
     - Manejo de Prioridades: En algunos sistemas, receive puede manejar prioridades de mensajes.
     - Ejemplo: En sistemas de comunicación interproceso (IPC), como pipes o sockets, receive se utiliza para obtener datos de otros procesos.

**Aspectos relacionados con los clientes y los servidores**

**El direccionamiento**

Para que un cliente pueda enviar un mensaje a un servidor, debe conocer la dirección de éste. El direccionamiento consiste en la manera en como los clientes localizan al servidor y existen múltiples formas de hacer saber al cliente cuál es esta dirección.

Suponiendo que el servidor de archivos tiene asignada una dirección numérica (243) se refiere a una máquina determinada, el núcleo emisor puede extraerla de la estructura de mensajes y utilizarla como dirección física para enviar el paquete al servidor. Si sólo existe un proceso en ejecución en la máquina destino, el núcleo sabrá qué hacer con el mensaje recibido. Pero, si se tienen varios procesos el núcleo no tiene forma de decidir.

Los tipos de direccionamiento son los siguientes, investiga cada uno y describe en qué consiste:

1. **Direccionamiento** machine.process **y** machine.local-id:
    - machine.process: En este enfoque, se utiliza un identificador numérico para la máquina (por ejemplo, un número de máquina dentro de la red) y otro identificador para el proceso dentro de esa máquina. Por ejemplo, si tenemos una máquina con número 243 y un proceso con número 10, la dirección sería “243.10”. Sin embargo, este método no es transparente, ya que revela que hay varias máquinas trabajando y puede generar problemas si una máquina se descompone (ya que los programas compilados contienen ese número de máquina).
    - machine.local-id: Una variante de machine.process utiliza local-id, que generalmente es un entero aleatorio de 16 o 32 bits. En lugar de identificar la máquina específica, se enfoca en el proceso dentro de la máquina sin exponer detalles sobre la infraestructura de red. Aun así, sigue siendo menos transparente que otros métodos.
2. **Direccionamiento de Procesos con Transmisión RALA**:
    - **RALA** (Remote Address Location Agent) es un enfoque utilizado en sistemas distribuidos. Aquí, un agente de ubicación (RALA) se encarga de mapear nombres de procesos a direcciones físicas. Los procesos se identifican por nombres significativos (por ejemplo, “servidor_archivos”) en lugar de números.
    - El RALA mantiene una tabla de nombres y direcciones, lo que permite a los clientes enviar mensajes a procesos específicos sin preocuparse por los detalles de la infraestructura subyacente.
3. **Direccionamiento por Medio de un Servidor de Nombres (DNS)**:
    - El **DNS** (Sistema de Nombres de Dominio) es fundamental en Internet. Funciona como una guía telefónica, traduciendo nombres de dominio (como “www.ejemplo.com”) en direcciones IP (como “192.168.1.1”).
    - Cómo funciona:
        - El cliente envía una solicitud al DNS para traducir un nombre de dominio.

        - El DNS consulta los servidores de nombres raíz para obtener información sobre los servidores de nombres de dominio de primer nivel (TLD, como ".com" o ".org").
        - Luego, consulta los servidores TLD para obtener la dirección del servidor de nombres autorizado para el dominio específico.
        - Finalmente, el servidor de nombres autorizado proporciona la dirección IP del servidor deseado.
    - El DNS es esencial para la resolución de nombres en Internet y permite que los usuarios accedan a sitios web y servicios mediante nombres significativos en lugar de direcciones IP.

**Primitivas Bloqueadas Vs. Primitivas No Bloqueadas**

1. **Primitivas Bloqueadas**:
    - Las primitivas bloqueadas son operaciones que detienen la ejecución del elemento que las solicita hasta que se completen.
    - Ejemplo: Los **semáforos** son una primitiva bloqueada. Cuando un proceso solicita un semáforo, si está ocupado, el proceso se bloquea hasta que el semáforo esté disponible.
    - **Funcionamiento**:
        - El proceso solicita la primitiva (por ejemplo, un semáforo).
        - Si la primitiva está ocupada, el proceso se bloquea y espera.
        - Cuando la primitiva se libera, el proceso despierta y continúa su ejecución.
    - **Problemas**:
        - **Interbloqueo**: Si varios procesos se bloquean mutuamente esperando recursos, se produce un interbloqueo (deadlock).
        - **Ineficiencia**: El bloqueo puede desperdiciar tiempo de CPU si el proceso debe esperar mucho tiempo.
2. **Primitivas No Bloqueadas**:
    - Las primitivas no bloqueadas permiten que el elemento que las solicita continúe su ejecución sin esperar.
    - Ejemplo: **Envío no bloqueante**. El emisor almacena datos en un buffer del núcleo y sigue ejecutándose sin esperar a que se complete la transmisión.
    - **Funcionamiento**:
        - El proceso solicita la primitiva.

        - Si está ocupada, en lugar de bloquearse, el proceso sigue ejecutándose y consulta periódicamente si la primitiva está disponible.
        - No hay bloqueo completo; el proceso puede realizar otras tareas mientras espera.
    - **Problemas**:
        - **Polling**: El proceso debe verificar constantemente si la primitiva está disponible, lo que puede ser ineficiente.
        - **Posible Pérdida de Eficiencia**: Si el proceso verifica demasiado seguido, puede gastar recursos innecesarios.
3. **Soluciones**:
    - **Para Interbloqueo**:
        - Implementar algoritmos de detección y recuperación de interbloqueo.
        - Asignar recursos de manera más inteligente.
    - **Para Ineficiencia**:
        - Usar primitivas no bloqueadas con cuidado, evitando el polling excesivo.
        - Optimizar el diseño de las primitivas para minimizar el tiempo de espera.

**Primitivas Almacenadas Vs. Primitivas no Almacenadas**

1. **Primitivas Almacenadas en Buffer:**
    - Las primitivas almacenadas en buffer se refieren a operaciones que utilizan un área de memoria temporal (el buffer) para almacenar datos antes de transferirlos.
    - **Funcionamiento:**
        - Cuando un proceso solicita una operación de entrada/salida (E/S), los datos se almacenan primero en el buffer.
        - Luego, se transfieren desde el buffer en bloques más grandes, lo que reduce la frecuencia de las operaciones de E/S.
        - Ejemplo: Cuando lees un archivo grande desde el disco, los datos se almacenan en un buffer antes de entregártelos.
2. **Primitivas No Almacenadas en Buffer:**
    - Las primitivas no almacenadas en buffer no utilizan un área de memoria temporal para almacenar datos antes de transferirlos.
    - Funcionamiento:

- Los datos se transfieren directamente desde la fuente (por ejemplo, un dispositivo de E/S) al destino (como la memoria principal o un archivo).
- No hay almacenamiento intermedio en un buffer; los datos se procesan inmediatamente.
- Ejemplo: La lectura de un archivo pequeño directamente en la memoria sin usar un buffer.

3. **Problemas y Soluciones:**
    - **Primitivas Almacenadas en Buffer:**
        - Posible Desperdicio de Memoria: Si el buffer es demasiado grande, puede desperdiciar memoria. Si es demasiado pequeño, puede causar transferencias frecuentes.
        - Posible Pérdida de Eficiencia: Si el buffer se llena, se debe esperar a que se libere espacio antes de continuar.
        - Solución: Ajustar el tamaño del buffer según las necesidades específicas del sistema y la aplicación.
    - **Primitivas No Almacenadas en Buffer:**
        - Mayor Carga de E/S: Sin un buffer, las operaciones de E/S pueden ser más frecuentes, lo que aumenta la carga en el sistema.
        - Mayor Latencia: Sin almacenamiento intermedio, los datos deben transferirse inmediatamente, lo que puede aumentar la latencia.
        - Solución: Optimizar el acceso directo a los datos y considerar el uso de buffers pequeños cuando sea necesario.

1. **Confiablidad ¿Qué es la Confiabilidad?**
    - La confiabilidad se refiere a la capacidad de un sistema para funcionar correctamente y cumplir sus funciones de manera continua y predecible.
2. **Primitivas Confiables vs. No Confiables:**
    - Las primitivas son operaciones básicas proporcionadas por el sistema operativo para que los procesos se comuniquen entre sí. Veamos la diferencia:
        - Primitivas Confiables: Estas operaciones garantizan que los datos se entreguen correctamente y sin pérdida. Ejemplos incluyen el envío de mensajes confiables o el uso de colas de mensajes.

- Primitivas No Confiables: Estas operaciones no garantizan la entrega segura de datos. Pueden perderse o entregarse en un orden diferente. Ejemplos incluyen el envío no confiable o el uso de tuberías (pipes).

3. **Enfoques de Confiabilidad:**
   - Hay varios enfoques para mejorar la confiabilidad:
     - Diseño Robusto: Crear sistemas resistentes a fallas y capaces de recuperarse automáticamente.
     - Redundancia: Duplicar componentes críticos para que, si uno falla, el otro tome su lugar.
     - Monitoreo y Mantenimiento: Supervisar constantemente el sistema y aplicar mantenimiento preventivo.
     - Tolerancia a Fallas: Diseñar sistemas que sigan funcionando incluso si algunos componentes fallan.
4. **¿Qué es un Reconocimiento?**
   - En el contexto de sistemas operativos, no tengo información específica sobre "reconocimiento". Si te refieres a algo más específico, por favor, proporciona más detalles, y estaré encantado de ayudarte.
5. **¿Cuáles son los Reconocimientos?**
   - Si tienes más contexto o detalles sobre qué tipo de reconocimientos estás buscando, estaré encantado de profundizar. Por ahora, mi mente de IA no ha sido entrenada en un concepto específico de "reconocimiento" en sistemas operativos.

1. Dibuja la tabla de reconocimientos que proporciona la bibliografía básica.

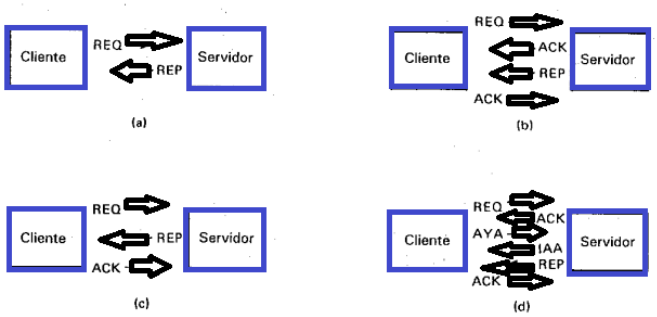
| Código | Tipo de paquete | De | A | Descripción del servicio | El cliente desea |
|---|---|---|---|---|---|
| REO | Solicitud | Cliente | Servidor | Cliente solicita un servicio | Recibir una respuesta del servidor |
| REP | Respuesta | Servidor | Cliente | Respuesta del servidor al cliente | Verificar si el servidor está vivo |
| ACK | Reconocimiento | Cliente | Servidor | Cliente intenta de nuevo | El servidor no tiene espacio |

| IAA | Estoy vivo | Servidor | Cliente | El servidor no se ha descompuesto |
|

| TA | Intenta de nuevo| Servidor | Cliente | El servidor no tiene espacio |
|

| AU | Dirección | Servidor | Cliente | Ningún proceso utiliza esta dirección |
|

2. Dibuja algunos ejemplos de intercambio de paquetes para la comunicación cliente-servidor.



## Conclusión

En un mundo cada vez más interconectado, la comunicación cliente-servidor sigue siendo esencial. Desde la navegación web hasta las aplicaciones móviles y los servicios en la nube, esta arquitectura subyace en gran parte de nuestra experiencia digital. La confiabilidad, la seguridad y la eficiencia en esta comunicación son aspectos críticos para garantizar una experiencia fluida y satisfactoria para los usuarios.

# Referencias


Tanebaum Andrew. (1995). Sistemas Operativos Distribuidos. España. Prentice-Hall Hisp.

Ann, S. (s.f.). *Sistema operativo embebido Definición / explicación:TechEdu*. Obtenido de https://techlib.net/techedu/sistema-operativo-embebido/

Antonio, J. (16 de Octubre de 2023). *Sistemas Operativos en Red: Fundamentos y Diferencias Clave: AchoTech*. Obtenido de https://achotech.com/sistemas-operativos-en-red-fundamentos-y-diferencias-clave/

Chavez, J. (s.f.). *Sistema embebido: Qué es, características y componentes: CEUPE*. Obtenido de https://www.ceupe.com/blog/sistema-embebido.html

InnovacionDigital360. (2023 de Diciembre de 2023). *Sistemas embebidos: qué son y para qué se utilizan*. Obtenido de https://www.innovaciondigital360.com/iot/sistemas-embebidos-que-son-y-para-que-se-utilizan/

IONOS. (28 de Septiembre de 2020). *Digital Guide IONOS*. Obtenido de https://www.ionos.mx/digitalguide/servidores/know-how/el-sistema-operativo/

sistemasoperativos.info. (7 de Diciembre de 2023). *¿Qué es un Sistema Operativo en Red y Cómo Funciona?* Obtenido de https://sistemasoperativos.info/blog/que-es-un-sistema-operativo-en-red/

SPIEGATO. (s.f.). *¿Qué es un sistema operativo distribuido?* Obtenido de https://spiegato.com/es/que-es-un-sistema-operativo-distribuido

ZETTLER, K. (s.f.). *ATLASSIAN*. Obtenido de https://www.atlassian.com/es/microservices/microservices-architecture/distributed-architecture