Tarea 5. Modelo cliente-servidor

SISTEMAS CONCURRENTES Y DISTRIBUIDOS

Ignacio David Vázquez Pérez

218292866

Objetivo:

Conocer los pormenores de uno de los modelos mas sencillos y eficientes, el modelo cliente-servidor.

Introducción:

La organización y comunicación en sistemas informáticos y de red son aspectos cruciales en la era digital en la que vivimos. Dos de los modelos fundamentales que rigen estas interacciones son el "Modelo Cliente-Servidor" y el "Modelo OSI" (Open Systems Interconnection). En esta tarea, exploraremos en detalle ambos modelos, examinando sus características, ventajas y aplicaciones en el mundo de la tecnología de la información.

Al comprender a fondo estos dos modelos, obtendremos una visión más clara de cómo funcionan los sistemas informáticos y las redes, lo que será fundamental para aquellos que deseen diseñar, administrar y optimizar sistemas de tecnología de la información en una amplia variedad de aplicaciones.

Actividades a realizar

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:**
  Un cliente es un proceso que solicita un servicio o recursos del sistema operativo o de otro proceso en la red. Los clientes suelen ser aplicaciones o usuarios que necesitan acceder a servicios proporcionados por los servidores.
  1. **Iniciadores de Solicitudes:** Los Clientes son los que inician solicitudes de servicios o recursos específicos. Pueden ser aplicaciones de usuario o componentes del sistema que requieren ciertas funciones.
  2. **Pasivos:** En general, los Clientes son pasivos en el sentido de que esperan respuestas a sus solicitudes. Envían solicitudes a los Servidores y luego esperan a que los Servidores procesen esas solicitudes y devuelvan los resultados.

3. **Interfaz de Usuario:** Los Clientes a menudo tienen una interfaz de usuario que permite a los usuarios interactuar con el sistema operativo y realizar acciones como iniciar aplicaciones, abrir archivos, etc.
4. **No Proveen Recursos:** Los Clientes generalmente no proveen servicios ni recursos a otros procesos. Su función principal es consumir servicios proporcionados por Servidores.

- **Servidores:**

  Un servidor es un proceso que proporciona servicios o recursos a los clientes. Está diseñado para escuchar las solicitudes de los clientes, procesarlas y enviar respuestas adecuadas.

  1. **Proveedores de Servicios:** Los Servidores son responsables de proporcionar servicios específicos o recursos a los Clientes. Estos servicios pueden incluir funciones como la gestión de archivos, la impresión, la autenticación, la base de datos, etc.
  2. **Esperan Conexiones:** Los Servidores están constantemente en espera de solicitudes entrantes de los Clientes. Cuando un Cliente envía una solicitud, el Servidor la recibe, la procesa y envía una respuesta de vuelta al Cliente.
  3. **Ejecutan Tareas en Segundo Plano:** A menudo, los Servidores operan en segundo plano y pueden ejecutarse de forma continua o bajo demanda, atendiendo múltiples solicitudes de Clientes.
  4. **Pueden Proveer Recursos a Varios Clientes:** Los Servidores pueden atender a múltiples Clientes simultáneamente, proporcionando servicios a varios usuarios o aplicaciones al mismo tiempo.

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

Las principales ventajas del modelo:

- Su sencillez.
- La eficiencia.
- La pila del protocolo es más corta y por tanto más eficiente.
- Si todas las máquinas fuesen idénticas, sólo se necesitarían tres niveles de protocolos.

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.

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

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 tipo de direccionamiento, los procesos en una red o sistema distribuido se identifican mediante dos componentes: el "machine" (máquina) y el "process" (proceso). El "machine" se refiere a la identificación de la máquina o nodo en la red, como una dirección IP o un nombre de host. El "process" se refiere a la identificación única del proceso en esa máquina.
- **Machine.Local-ID:** Similar al anterior, pero aquí, el "machine" se refiere a la identificación local de la máquina en un contexto particular. Por lo tanto, se utiliza principalmente para la comunicación dentro de una máquina o nodo específico.

2. Direccionamiento de procesos con transmisión RALA de Procesos.

- **RALA (Remote Address Location Assignment):** Este enfoque se utiliza en sistemas distribuidos para asignar direcciones a procesos en función de su ubicación remota. Los procesos pueden solicitar una dirección de RALA cuando necesitan comunicarse con un proceso remoto.
- **Proceso con Transmisión RALA:** En este contexto, el direccionamiento se basa en la asignación de direcciones RALA a procesos que necesitan comunicarse de forma remota. Los procesos envían mensajes utilizando estas direcciones RALA, y el sistema de direccionamiento se encarga de enrutar los mensajes al proceso remoto correspondiente.

3. Direccionamiento por medio de un servidor de nombres.

Este tipo de direccionamiento simplifica la administración y la escalabilidad en sistemas distribuidos, ya que permite la flexibilidad de cambiar las direcciones o ubicaciones de los recursos sin afectar a los usuarios o las aplicaciones que utilizan nombres.

- **Servidor de Nombres:** En este enfoque, se utiliza un servidor de nombres o un servicio de resolución de nombres para asignar y recuperar direcciones o nombres a objetos o recursos en la red. Los clientes envían solicitudes al servidor de nombres para obtener la dirección de un recurso específico.
- **Direccionamiento por Nombre:** Los procesos o recursos se identifican mediante nombres significativos en lugar de direcciones numéricas o identificadores. Los usuarios y aplicaciones

pueden utilizar estos nombres para acceder a recursos sin necesidad de conocer las direcciones numéricas.

Primitivas Bloqueadas Vs. Primitivas No Bloqueadas

1. ¿Qué son las primitivas bloqueadas y cómo funcionan?

- **Definición:** Las primitivas bloqueadas son operaciones que pueden bloquear un hilo o proceso hasta que ciertas condiciones se cumplan. Estas operaciones pueden incluir bloquear un recurso compartido, como un semáforo o un mutex, o esperar a que ocurra un evento específico antes de continuar.
- **Funcionamiento:** Cuando un hilo o proceso intenta adquirir un recurso bloqueado o realizar una operación bloqueante, puede quedar en espera hasta que ese recurso se libere o se cumplan las condiciones necesarias. Durante este tiempo, el hilo o proceso no puede ejecutar ninguna otra tarea y queda inactivo.

2. ¿Qué son las primitivas no bloqueadas y cómo funcionan?

- **Definición:** Las primitivas no bloqueadas son operaciones que garantizan que al menos un hilo o proceso pueda hacer progreso, incluso en situaciones de concurrencia. Esto significa que, aunque varios hilos puedan estar compitiendo por recursos, ninguno quedará permanentemente bloqueado.
- **Funcionamiento:** Las primitivas no bloqueadas se diseñan de manera que, si un hilo no puede completar una operación debido a la competencia por recursos, este hilo se reorganiza y vuelve a intentar la operación hasta que tenga éxito. Otros hilos pueden avanzar mientras tanto.

3. ¿Qué problemas tiene cada una y como se puede soluciona (si es que se puede)?

- **Primitivas Bloqueadas:**
  - **Problemas:** Los problemas comunes asociados con las primitivas bloqueadas incluyen el riesgo de bloqueo mutuo (dos o más hilos bloqueándose entre sí), la posibilidad de que un hilo quede en espera indefinidamente (deadlock) y una menor eficiencia en situaciones de concurrencia con alta competencia por recursos.
  - **Soluciones:** Para mitigar estos problemas, se pueden utilizar estrategias como la asignación cuidadosa de recursos, el uso de timeouts (límites de tiempo) para liberar recursos bloqueados después de un tiempo determinado, y el diseño cuidadoso de algoritmos de bloqueo para evitar bloqueos mutuos.
- **Primitivas No Bloqueadas:**
  - **Problemas:** Aunque las primitivas no bloqueadas evitan problemas como el bloqueo mutuo y el deadlock, pueden ser más difíciles de diseñar e implementar correctamente. Además, pueden resultar en un mayor uso de recursos del sistema debido a los intentos repetidos.
  - **Soluciones:** Para trabajar con primitivas no bloqueadas, es importante diseñar algoritmos cuidadosamente y asegurarse de que siempre haya al menos un hilo que haga progreso.

Además, se pueden utilizar mecanismos como exponer tiempos límite para operaciones o implementar estrategias de retroceso exponencial para evitar congestiones excesivas.

Primitivas Almacenadas Vs. Primitivas no Almacenadas

1. ¿Qué son las primitivas almacenadas en buffer y cómo funcionan?

- **Definición:** Las primitivas almacenadas en buffer son mecanismos de sincronización y comunicación en los que los datos o mensajes se almacenan temporalmente en un buffer o cola antes de ser consumidos por otro proceso. Estas primitivas incluyen estructuras de datos como colas, buffers circulares y canales de comunicación que almacenan datos hasta que se lean o procesen.
- **Funcionamiento:** Cuando un proceso quiere enviar datos o mensajes a otro, los coloca en el buffer o cola. El proceso receptor los recoge y procesa cuando esté listo. Esto permite que los procesos funcionen de manera asincrónica y evita bloqueos.

2. ¿Qué son las primitivas no almacenadas en buffer y cómo funcionan?

- **Definición:** Las primitivas no almacenadas en buffer son mecanismos de sincronización y comunicación en los que no se almacenan datos en un buffer intermedio. Los datos se transfieren directamente entre los procesos emisor y receptor, y la comunicación es más eficiente en términos de memoria y latencia.
- **Funcionamiento:** En este enfoque, el proceso emisor envía datos directamente al proceso receptor cuando está listo para hacerlo. No hay almacenamiento temporal en un buffer. Esto significa que el proceso emisor debe esperar hasta que el proceso receptor esté listo para recibir los datos.

3. ¿Qué problemas tiene cada una y como se puede solucionar (si es que se puede)?

- **Primitivas Almacenadas en Buffer:**
  - **Problemas:** Uno de los problemas potenciales de las primitivas almacenadas en buffer es el desbordamiento o la falta de espacio en el buffer si los procesos no pueden consumir los datos lo suficientemente rápido. Esto puede llevar a la pérdida de datos o al bloqueo del proceso emisor si se espera que el buffer se vacíe antes de continuar.
  - **Soluciones:** Para evitar el desbordamiento, se pueden implementar mecanismos como el ajuste del tamaño del buffer según la carga del sistema o el uso de estrategias de "drop tail" para eliminar datos excedentes cuando el buffer está lleno. Además, se pueden utilizar técnicas de programación que eviten bloqueos en el proceso emisor si el buffer se llena.
- **Primitivas No Almacenadas en Buffer:**
  - **Problemas:** El principal problema de las primitivas no almacenadas en buffer es que pueden generar bloqueos si el proceso emisor intenta enviar datos antes de que el proceso receptor esté preparado para recibirlos. Esto puede conducir a situaciones de espera indefinida.

- **Soluciones:** Para evitar bloqueos, se pueden utilizar técnicas como el uso de señales o eventos para indicar al proceso emisor que el proceso receptor está listo para recibir datos. También se pueden implementar protocolos de comunicación cuidadosamente diseñados que eviten bloqueos y garanticen la sincronización adecuada entre los procesos.

Confiablidad

1. ¿Qué es la confiabilidad?

La confiabilidad se refiere a la capacidad de un sistema, componente o proceso para funcionar correctamente y de manera consistente bajo diversas condiciones y durante un período de tiempo determinado. En el contexto de sistemas informáticos y de comunicación, la confiabilidad implica que estos sistemas deben ser capaces de ofrecer servicios de manera constante y sin fallos significativos.

2. ¿Cuáles son las primitivas confiables y cuales las no confiables?

- **Primitivas Confiables:** Las primitivas confiables se refieren a operaciones o protocolos que se diseñan de manera que se garantiza la entrega y la integridad de los datos. Esto significa que se toman medidas para asegurarse de que los datos enviados sean recibidos correctamente, sin errores ni pérdidas.
- **Primitivas No Confiables:** Las primitivas no confiables son operaciones o protocolos en los que no se garantiza la entrega y la integridad de los datos. En estos casos, se asume que pueden ocurrir errores o pérdida de datos y no se toman medidas para su corrección.

3. ¿Cuáles son los diferentes enfoques que trata la confiabilidad?

- **Redundancia:** La redundancia implica duplicar componentes críticos en un sistema para que, si uno de ellos falla, otro pueda tomar su lugar y mantener la operación sin interrupciones.
- **Detección y corrección de errores:** Utilizar técnicas para detectar y corregir errores en los datos transmitidos o almacenados, como códigos de corrección de errores.
- **Tolerancia a fallos:** Diseñar sistemas que puedan continuar funcionando incluso cuando se produzcan fallos en componentes individuales.
- **Planificación y mantenimiento preventivo:** Implementar planes de mantenimiento regular y procesos de monitoreo para anticipar y prevenir fallos.

4. ¿Qué es un reconocimiento?

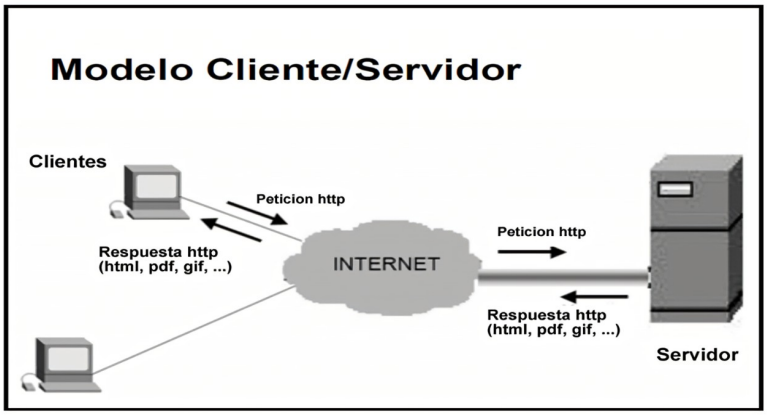
Un reconocimiento es una señal o mensaje enviado por el receptor de un mensaje o paquete de datos para confirmar que ha recibido correctamente el mensaje. Es una parte importante de los protocolos de comunicación confiables, ya que permite al remitente saber que la información ha sido entregada con éxito.

5. ¿Cuáles son los reconocimientos?

Los reconocimientos son mensajes o señales específicas que se utilizan para confirmar la recepción exitosa de datos. Algunos ejemplos comunes de reconocimientos incluyen:

- **ACK (Acknowledgment):** Un mensaje de confirmación que indica que el receptor ha recibido y procesado correctamente un paquete de datos.
- **NACK (Negative Acknowledgment):** Un mensaje que indica que se ha producido un error en la recepción o procesamiento de un paquete de datos, y que se solicita una retransmisión.
- **SYN-ACK (Synchronize-Acknowledge):** Utilizado en el proceso de establecimiento de una conexión en el protocolo TCP/IP.

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



1. El Cliente (una aplicación en una computadora) desea obtener una página web alojada en un Servidor Web remoto.
2. El Cliente inicia una solicitud HTTP (por ejemplo, una solicitud GET) para la página web y envía esta solicitud al Servidor.
3. La solicitud del Cliente se empaqueta en un paquete de datos que contiene la dirección IP del Servidor y el puerto asociado al servicio web (por ejemplo, el puerto 80 para HTTP).
4. El paquete de datos se envía a través de la red a través de varios nodos de enrutamiento intermedios.
5. El paquete llega al Servidor Web remoto, que está escuchando en el puerto 80.
6. El Servidor Web procesa la solicitud, recupera la página web solicitada y la empaqueta en una respuesta HTTP.
7. La respuesta del Servidor se empaqueta en un paquete de datos y se envía de regreso al Cliente a través de la red.
8. El paquete llega al Cliente, que lo desempaqueta y procesa la respuesta HTTP.
9. La página web solicitada se muestra en el navegador web del Cliente.

Conclusión

El modelo cliente-servidor es una estructura fundamental en la organización y comunicación de sistemas informáticos y de red. En este modelo, los clientes son procesos que solicitan servicios o

recursos, mientras que los servidores son procesos que proporcionan esos servicios o recursos.

El modelo cliente-servidor utiliza un protocolo "requerimiento-respuesta" sin conexión para evitar costos excesivos en la comunicación. Sus principales ventajas incluyen la sencillez, la eficiencia y una pila de protocolo más corta.

En comparación con el modelo OSI, el modelo cliente-servidor se destaca por su simplicidad y eficiencia en la comunicación. El direccionamiento en este modelo puede realizarse de varias maneras, como el direccionamiento "machine.process" y "machine.local-id", el direccionamiento de procesos con transmisión RALA y el direccionamiento por medio de un servidor de nombres, que permite asignar y recuperar direcciones o nombres de recursos en la red.

Las primitivas bloqueadas pueden generar problemas como el bloqueo mutuo y el deadlock, pero se pueden abordar mediante una asignación cuidadosa de recursos y el uso de timeouts. Las primitivas no bloqueadas evitan estos problemas pero requieren un diseño más cuidadoso y pueden utilizar señales o eventos para la sincronización.

En cuanto a la confiabilidad, se pueden aplicar enfoques como la redundancia, la detección y corrección de errores, la tolerancia a fallos y el mantenimiento preventivo para garantizar el funcionamiento confiable de los sistemas.

Por último, los reconocimientos son mensajes cruciales en la comunicación confiable, como ACK para confirmar la recepción exitosa y NACK para indicar errores, que desempeñan un papel fundamental en protocolos como TCP/IP.

Bibliografía

- Tanebaum Andrew. (1995). Sistemas Operativos Distribuidos. España. Prentice-Hall Hisp.
- Mclver Ann. (2011). Sistemas Operativos. México. Cengage Learning.
- Tanenbaum, A.,& Van Steen M.(2008). Sistemas Distribuidos, Principios y Paradigmas. (Segunda ed.). Prentice Hall.
- Tanenbaum, A.(2011). Redes de Computadoras. (Quinta ed.). Prentice Hall.
- Elmasri, R., Gil Carrick, A., & Levine, D. (2010). Sistemas Operativos, Un enfoque en espiral. McGraw--Hill.
- Infranetworking. (s.f.). Modelo cliente-servidor. https://blog.infranetworking.com/modelo-cliente-servidor/
- Muy Tecnológicos. (s.f.). Arquitectura cliente-servidor. https://www.muytecnologicos.com/diccionario-tecnologico/arquitectura-cliente-servidor
- IONOS. (s.f.). ¿Qué es el modelo cliente-servidor? Pros y contras. https://www.ionos.mx/digitalguide/servidores/know-how/modelo-cliente-servidor/
- CCM. (s.f.). ¿Qué es la arquitectura cliente-servidor y cómo funciona? https://es.ccm.net/aplicaciones-e-internet/museo-de-internet/enciclopedia/10682-que-es-la-

arquitectura-cliente-servidor-y-como-funciona/

- CIDECA. (s.f.). 4.2. Arquitectura cliente/servidor. http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro21/42_arquitectura_clienteservidor.html
- IberAsync. (s.f.). Arquitectura Cliente/Servidor: modelo de 3 capas. https://iberasync.es/arquitectura-cliente-servidor-modelo-de-3-capas/
- Jiménez, L. (s.f.). La arquitectura cliente-servidor: interacción entre sistemas. https://leojimzdev.com/la-arquitectura-cliente-servidor-interaccion-entre-sistemas/