TEMA 8
COMUNICACION EN LOS SISTEMAS DISTRIBUIDOS

8.1 INTRODUCCION A LA COMUNICACION EN LOS SISTEMAS DISTRIBUIDOS: la diferencia entre un sistema distribuido y un sistema de un único procesador es la comunicación entre los procesos. En un sistema de un único procesador es la comunicación entre los procesos. En un sistema de un único procesador la comunicación es mediante la existencia de la memoria compartida. En un sistema distribuido no existe la memoria compartida. Los procesos para comunicación se operan a reglas "PROTOCOLOS". Estos protocolos en un área amplia toman la forma de varias capas cada una con su meta y reglas. Los mensajes se intercambian de muchas formas. (Ej. llamadas a un procedimiento remoto).

8.2 PROTOCOLOS CON CAPAS: Debido a la ausencia de memoria compartida, toda la comunicación se basa en la *transferencia de mensajes*.

- Este modelo **OSI (modelo de referencia para interconexión de sistemas abiertos)** esta diseñado para permitir la comunicación de los **sistemas abiertos** (con los preparados para comunicarse con otro sistema abierto mediante *reglas estándar);* se establece el formato, contenido y significados de los mensajes recibidos y enviados, constituyen los PROTOCOLOS (son acuerdos en el formato que debe desarrollarse la comunicación).

El modelo OSI distingue dos tipos de **protocolos**:

- **Orientados hacia las conexiones:** antes de intercambiar los datos, el emisor y el receptor establece en forma explícita de antemano la conexión, se dice el protocolo al usar y al terminar su conexión. (Ej. teléfono).
- **Sin conexión:** no se configura de antemano, el mismo transmite el primer mensaje cuando está listo (Ej. depositar una carta en un buzón).

Cada capa proporciona una interfaz con la otra capa por encima de ella, esta interfaz es un conjunto de operaciones que define el servicio que las capas ofrecen a sus usuarios. Cada protocolo utiliza la información de su capa. Cada protocolo de capa se puede cambiar, confiere gran flexibilidad.

8.3 INTRODUCCION AL MODELO CLIENTE - SERVIDOR (C - S): El modelo de la OSI tiene un problema: el costo, y el envío de mensaje. El modelo OSI no estructurado al sistema distribuido; lo hace el modelo c-s (cliente/servidor) como un grupo de procesos en cooperación que ofrecen servicios a los usuarios "SERVIDORES" y un grupo de procesos usuarios "CLIENTES". Este modelo se basa en un protocolo solicitud/respuesta; es sencilla y sin conexión, y no como OSI o TCP/IP (orientado a la conexión).
El cliente envía un mensaje de solicitud el servidor pidiendo un servicio. El servidor ejecuta el requerimiento y regresa los datos solicitados o un código de error. No se establece una conexión seno hasta que se lo solicite. Las capas de protocolo son más cotosas y más eficientes.

8.4 DIRECCIONAMIENTO EN C – S: Para que un cliente pueda enviar un mensaje a un servidor, debe conocer la dirección de este. Algunos esquemas de direccionamiento se basa en:

- La dirección de la máquina donde se envía el mensaje.
- *Identificar los procesos* destinatarios en vez de a las maquinas.
- Se asigna a cada proceso una única dirección que no tiene número de máquina.
- Se deja que cada proceso elija su propio identificador en un espacio de direcciones grande y disperso.
- Se utiliza hardware especial cada proceso elige su dirección en forma aleatoria.

8.5 PRIMITIVAS DE BLOQUEO VS. NO BLOQUEO EN C – S: Las primitivas de transferencia de mensajes son las **primitivas de bloqueo o primitivas sincronas:** El proceso emisor se suspende (se bloquea) mientras se envía el mensaje. El proceso receptor se suspende mientras se recibe el mensaje.
Otras son las **primitivas sin bloqueo o primitivas asíncronas:** el proceso emisor no se suspende, continua su cómputo paralelamente con la transmisión del mensaje, pero no se sale cuando termina la transmisión para poder utilizar el buffer de forma segura.
Generalmente se considera que las desventajas de las primitivas asíncronas no compensan las ventajas del máximo paralelismo que permiten lograr. A las primitivas de envío se las conoce como SEND y a los de receptores RECEIVE.

8.6 PRIMITIVAS ALMACENADAS EN BUFFER VS. NO ALMACENADAS: Las primitivas no almacenadas significan que una dirección se refiere a un proceso específico. Se puede mantener el mensaje un instante en el núcleo (Ej. para mensajes que llegan antes), de esta manera se intenta que los mensajes no se pierdan.
Las primitivas con almacenamiento en buffer utilizan buzón que es una estructura de datos donde se colocan los mensajes que llegan con una dirección perteneciente a un proceso específico. El problema que los buzones son infinitas y se pueden llenar, por esto el núcleo tendrá que decidir si mantener el mensaje por un momento o descártalo. Una solución consistirá en no dejar que un proceso envíe un mensaje si necesita espacio para su almacenamiento en el destino.

8.7 PRIMITIVAS CONFIABLES VS. NO CONFIABLES: se supone que el mensaje enviado será siempre recibido pero se puede perder por muchas causas cuando un cliente envía un mensaje se lo suspende hasta que el mensaje ha sido enviado, pero no hay garantía que el mensaje se ha entregado.
Una solución consistirá en volver a definir la semántica del SEND para hacerlo no confiable, de esta manera el sistema no garantiza la entrega del mensaje, sino que lo deja en manos del usuario. Otro método exige que el núcleo de la maquina receptora envíe un reconocimiento al núcleo de la maquina emisora. Otra solución sería que la misma respuesta funcione como un reconocimiento.

8.8 IMPLANTACION DEL MODELO C – S: Las principales opciones de diseño son:
° Direccionamiento: número de máquina, dirección de procesos entre otros.
° Bloqueo: primitivas con bloqueo. con copia al núcleo o con interrupciones.
° Almacenamiento en buffer: no usar el almacenamiento en buffer, descartar los mensajes inesperados. mantener temporal de los mensajes inesperados o Buzones.
° Confiabilidad: No confiable. //Solicitud - reconocimiento - respuesta - reconocimiento. //Solicitud - respuesta - reconocimiento.

Se combinan todas estas formando diseños distintos.
Otro aspecto de la implementación es el protocolo subyacente utilizado en la comunicación C-S.
Los principales tipos de paquetes son los siguientes:

- **REQ:** Solicitud de cliente a servidor por un servicio.
- **REP:** Respuesta de servidor a cliente.
- **ACK:** Reconocimiento de cualquiera de ellos a algún otro.
- **AYA**: ¿estas vivo?. De cliente a servidor.
- **IAA:** Estoy vivo. De servidor a cliente.
- **TA:** Intenta de nuevo. De servidor a clientes. No hay espacio.
- **AU**: Dirección desconocida. de servidor a cliente.

8.9 LLAMADA A UN PROCEDIMIENTO REMOTO (RPC): es una forma de intercambiar mensajes en un sistema distribuido.
El problema del modelo C-S es que la comunicación es mediante la E/S con los procedimientos SEND/RECEIVE que realizan la E/S.
Otra alternativa fue permitir a los programas que llamasen a procedimiento localizado en otras máquinas.
Cuando un proceso A que llama un procedimiento en otra máquina B se suspende, se ejecuta el procedimiento en esta máquina B. la información se transporta de un lado otro mediante parámetros y regresa en el resultado del procedimiento.
El programador no se preocupa de una transferencia o de la E/S; a esto se llama LLAMADA A PROCEDIEMITNO (REC). El procedimiento que hace la llamada y el que recibe se ejecutan en máquinas diferentes.

8.10 OPERACION BASICA DE RPC: Una llamada convencional a un procedimiento (que opera en una sola máquina) es: el programa llamador se carga en la memoria, después que la lectura termine su ejecución, se coloca el valor de regreso en un registro, se elimina la dirección de regreso, se transfiere de nuevo el control al llamador y éste determina los parámetros de la pila y regresa a su estado original.
La idea es que la RPC se aparezca lo mas posible a una llamada (ser transparente). El procedimiento que hace la llamada no debe ser conciente de que el procedimiento llamado se ejecuta en otra maquina o viceversa.
Pasos de una RPC:

- El procedimiento cliente llama al STUB del cliente de la manera usual.
- El STUB del cliente construye un mensaje y hace un señalamiento al núcleo.

- El núcleo envía el mensaje al núcleo remoto.
- El núcleo remoto proporciona el mensaje al STUB del servidor.
- El STUB del servidor desempaca los parámetros y llama al servidor.
- El servidor realiza el trabajo y regresa el resultado al STUB.
- El STUB del servidor empaca el resultado en un mensaje y hace un señalamiento al núcleo.
- El núcleo remoto envía el mensaje al núcleo del cliente.
- El núcleo del cliente da el mensaje al STUB del cliente.
- El STUB desempaca el resultado y regresa al cliente.

De esta manera se ha convertido la llamada local del procedimiento cliente al STUB del cliente, en una llamada local al procedimiento servidor.

8.11 TRANSFERENCIA DE PARAMETROS EN RPC: El empacamiento de parámetros en un mensaje se llama **ordenamiento de parámetros.** El mensaje también contiene el nombre o nº del procedimiento por llamar. Cuando el mensaje llega al servidor el resguardo STUB le examina para ver cuando el procedimiento necesita y lo lleva a cabo. Los elementos del mensaje son verificados del procedimiento y parámetros.

8.12 CONEXION DINAMICA (DYNAMIC BINDING) EN RPC: El tema es la forma en que el cliente localiza al servidor.
Un método consiste en integrar dentro del código del cliente la dirección (en la red) del servidor. El problema es que si se cambia el servidor se tiene que volver a compilar el programa.
Una solución es la **conexión dinámica** para que concuerden los clientes y los servidores.
Se inicial especificando al servidor, cuando el servidor inicia su ejecución envía un mensaje a un programa CONECTOR para darle a conocer su existencia (registro del servidor).
El servidor puede cancelar su registro con el conector si no va a prestar servicio.
El cliente localiza al servidor enviando un mensaje al conductor especificando lo que necesita. Cuando el cliente llama a un procedimiento por primera vez si existe un servidor adecuado al conector puede ser un cuello de botella para muchas cargas de trabajo.

8.13 SEMANTICA DE RPC EN PRESENCIA DE FALLOS: El objetivo de RPC es ocultar la comunicación al hacer que las llamadas a procedimientos remotos se parezcan a las llamadas locales.
El problema es cuando aparecen los errores porque las diferencias entre las llamadas locales y remotas no son fáciles de encubrir.
Situaciones:
° **El cliente no puede localizar al servidor.** Puede estar inactivo el servidor, puede utilizar nuevas versiones incompatibles con el resguardo del cliente.
° **Perdida de mensajes de solicitud.** El núcleo inicializa un cronometro al enviar la solicitud. Si el tiempo termina y no regresa una respuesta se vuelve a enviar el mensaje, o puede que el servidor este inactivo.
° **Perdida de mensajes de respuesta.** Se utiliza un cronometro, si no llega una respuesta se vuelve a enviar la solicitud, pero el núcleo del cliente no sabe porque no hubo respuesta. Si incrementa las solicitudes para que el núcleo del servidor pueda diferenciar entre original y retransmisión.
° **Fallos del servidor.** Antes de recibir una solicitud (inactivo) o antes de disponer de una solicitud. Soluciones para que el núcleo distinga entre estas dos:
Semántica al menos una: esperar hasta que el servidor vuelva a arrancar o intente realizar de nuevo las operaciones. Garantiza que las RPC se ha realizado al menos una vez, pero es posible que se realice mas veces.
Semántica a lo mas una: no se reintenta y se informa del fallo. Garantiza que la RPC se realiza a lo mas una vez, pero es posible que no se realice ni una vez.
Semántica de no garantizar nada: cuando el servidor falla, el cliente no obtiene ayuda. La RPC se puede realizar un número de veces que va desde "o" hasta "n".
Semántica de exactamente una: Es la solución deseable pero generalmente no existe forma de garantizar esto.
Aquí se diferencian los sistemas con un único procesador y los sistemas distribuidos porque con un único procesador si falla el servidor falla el cliente y la recuperación no es posible ni necesaria, pero con sistemas distribuidos si es posible y necesario.
° **Fallos del cliente:** se genera una solicitud de trabajo o cómputo que al fallar ya nadie espera, "HUERFANO". Esto genera problemas como desperdicios de ciclo de CPU, bloqueo de archivos, apropiación de recursos.
Soluciones a cómputos huérfanos.

Exterminación. Se crea un registro que indica lo que va a hacer el resguardo del cliente antes de que emita la RPC. Luego del re arrancar se elimina el huérfano. Hay sobrecarga de E/S.

Reencarnación: se divide el tiempo en épocas. Cuando un cliente re arranca envía un mensaje a todos las máquinas declarando el inicio de una nueva época. Al recibir este mensaje se elimina todos los cómputos remotos.

Reencarnación sutil: cuando llega un mensaje de cierta época, la máquina verifica si tiene cómputos remotos, si no se localiza al poseedor se elimina el cómputo.

Expiración: si cada RPC se le da un tiempo para realizar su trabajo, si pasa se pasa el tiempo antes de re arrancar todos los huérfanos desaparecidos.

8.14 ASPECTOS DE LA IMPLANTACION EN RPC: el desempeño o performance es fundamental en los sistemas distribuidos y depende de la velocidad de la comunicación y la velocidad depende de la implantación.

- **Protocolos RPC.** Se elige entre un protocolo *orientado a la conexión* o un *protocolo sin conexión* entre cliente y servidor.

Otro aspecto importante es longitud del paquete y el mensaje:

° Realizar una RPC tiene un alto costo por la cantidad de datos que se envían.

° El protocolo y la red deben permitir transmisiones largas para minimizar el mínimo de RPC.

Otra opción es el protocolo estándar de propósito general o alguno especifico en RPC, como el IP:

- **Reconocimientos.** Cuando los RPC son grandes se deben dividir en muchos paquetes pequeños, surge el problema del reconocimiento. Los paquetes se reconocerán individualmente o grupalmente. Estrategias para reconocer:
    - protocolo detenerse y esperar: enviar y esperar el reconocimiento, luego enviar el otro.
    - Protocolo de chorro: envía todo como puede y se reconoce todo el conjunto.

  Cuando el paquete elige a un receptor que no lo puede aceptar hay un ERROR DE EJECUCIÓN y el paquete se pierde.
- **Ruta critica.** Es la serie de instrucciones que se ejecutan con cada RPC. Es importante determinar en que parte de la ruta crítica se ocupa la mayor parte del tiempo que demanda la RPC, que depende del tiempo de RPC y la cantidad de datos.

En RPC con transporte mínimo la mayor parte del tiempo se ocupa en el cambio de contrato, en RPC con transporte de 1Kb o más la mayor parte del tiempo se ocupa en el tiempo de transmisión.

- **Copiado:** El número de veces que se debe copiar un mensaje varía según el hardware, software y el tipo de llamada.
- **Manejo del cronometro:** Se inicializa al enviar un mensaje y se espera una respuesta, si la respuesta no llega se re transmite el mensaje. Para el establecimiento de un cronómetro se requiere construir una estructura de datos que especifique el momento en que el cronómetro debe detener y la acción a realizar, cuando eso no sucede.

8.15 AREAS DE PROBLEMAS EN RPC: La RPC mediante el modelo C-S se utiliza como base de los s. O. Distribuidos. La idea es que la RPC sea ***transparente****:* pero la mayoría no cumplen. Algunos de los problemas son:

- **Variables globales**: no permiten la transparencia en la implementación.
- **Lenguajes débilmente tipificados**: Ej. “c”, no se puede determinar su tamaño.
- No siempre se decide el tipo de parámetros.

8.16 COMUNICACION EN GRUPO: La comunicación de RPC no es solo entre dos procesos (cliente/servidor), a veces es entre varios procesos. Por ejemplo, un grupo de servidores de archivos ofrecen un único servicio de archivo tolerante a fallos. El cliente envía un mensaje a todos los servidores para garantizar la ejecución de la solicitud aunque alguno falle. La RPC no puede controlar la comunicación de un servidor con muchos receptores, a menos que realice RPC con cada uno en forma individual.

Un grupo es una colección de procesos que actúan junto en un sistema o de alguna forma determinada por el usuario.

La propiedad fundamental de todos los grupos es que cuando un mensaje se envía al propio grupo, todos los miembros lo reciben.

La comunicación es UNO-MUCHOS (un emisor, muchos receptores). Los grupos son dinámicos, se crean y se destruyen; un proceso se une a un grupo o deja a otro, un proceso puede ser miembro de varios grupos a la vez.

Implementar la comunicación en grupo depende del hardware y se llama “MULTITRANSMISIÓN”: cuando las máquinas que responden a la dirección.

Otra es la **transmisión simple** para redes que no soportan multitransmisión. Significa que los paquetes se entregan a todas las máquinas y cada una debe verificar si va a dirigirse e ella o lo descarta.

Otra solución es implantar la comunicación en grupo por **paquetes individuales** a c / u de los miembros del grupo por parte del emisor, que es la **uní transmisión**.

8.17 ASPECTOS DEL DISEÑO DE LA COMUNICACION EN GRUPO: Existen posibilidades como:

- Almacenamiento en buffers vs. El no almacenamiento.
- Bloqueo vs. No bloqueo.

Grupos cerrados vs. grupos abiertos. En los **grupos cerrados s**olo los miembros del grupo se envían mensajes entre ellos, los extraños solo pueden enviar a un miembro de grupo en la individual.
En los **grupos abiertos c**ualquier proceso del sistema puede enviar a cualquier grupo.
El grupo cerrado se usa para el procesamiento paralelo. (Ej. un grupo de procesos que trabajan de manera conjunta, tienen su propio objetivo y no interactúan con el mundo exterior).

Grupos de compañeros vs. Grupos jerárquicos. En los GRUPOS DE COMPAÑEROS todos los procesos son iguales, no hay jerarquías. Es simétrico y no tiene un punto de fallo, el grupo puede continuar aunque algunos de los procesos fallen. Al tomar decisiones hay retrasos por la comunicación entre todos.
En los GRUPOS JERARQUICOS un proceso es coordinador y todos los demás trabajadores, cualquier solicitud que se genere externamente o en un trabajador se envía al coordinador y decide cual trabajador es el más adecuado para llevarlo a cabo y se lo envía. La pérdida del coordinador paraliza el grupo. En condiciones normales, toma decisiones sin molestar a los demás procesos.

Membresía del grupo. La comunicación en grupo requiere crear y eliminar grupos. Unir y separar procesos a grupos. Se puede tener un **servidor de grupos** al cual enviar todas las solicitudes, es fácil y eficiente; para la **administración centralizada** de grupo representa un punto de fallo.
Otra posibilidad es la **administración distribuida** de las membresías de grupo, para anunciar un proceso su presencia a un grupo, o al dejarlo debe enviar un mensaje, para esto la E/S al grupo debe sincronizarse.
A veces faltan tantas máquinas que el grupo no puede funcionar, se necesita un protocolo para reconstruir al grupo, algún proceso tomará la iniciación. El protocolo resolverá la situación si dos o mas procesos quieren al mismo tiempo reconstruir el grupo.

Direccionamiento al grupo. Los grupos deben poder direccionarse, al igual que los procesos. A cada proceso se le da una dirección única, o se le pide al emisor una lista de los destinos (Ej. lista de direcciones IP), pero no es transparente.
Otro método es el direccionamiento de predicados, el mensaje se envía al grupo que contiene un predicado a evaluar, si es verdadero se acepta el mensaje y si es falso se descarta el mensaje.

Primitivas Send Y Receive. El envío de un mensaje a un grupo no puede ser una llamada a un procedimiento ya que con la comunicación en grupo existirá en potencia *n* respuestas diferentes. Se utilizan llamadas explicitas para el envío y recepción (modelo de un solo sentido). Uno de los parámetro que indica el destino y el segundo apunta al mensaje por enviar.

Atomicidad. Los sistemas de comunicación en grupo están diseñados para que los mensajes enviados al grupo lleguen correctamente a todos los miembros o a ninguno de ellos.
La propiedad de "todo o nada" en la entrega se llama **atomicidad** o **transmisión atómica.** Facilita la programación de los sistemas distribuidos. La única forma de garantizar de que cada destino recibió todos sus mensajes es que envíe de regreso su reconocimiento al recibir.

Ordenamiento de mensajes. El ordenamiento de mensajes para el envío de mensajes en la comunicación en grupo es importante. La mejor garantía es la entrega inmediata de todos los mensajes en el orden en que se enviaron.

Grupos traslapados. Un proceso puede ser miembro de varios grupos a la vez. Se puede generar cierta inconsistencia. El problema es que Existe un tiempo global dentro de cada grupo y No existe coordinación entre varios grupos.

Escalabilidad. Se debe considerar como funcionaran los algoritmos cuando:

- Hay Grupos con centenas o miles de miembros.
- Centenas o miles de grupos.

- Utilización de varias LAN y compuertas conectadas
- Diseminación de grupos en varios continentes.