CAPITULO 21

MODELO CLIENTE SERVIDOR

21.1. Introducción [10][11][S.O.]

Como vimos en el modelo ISO son necesarias siete capas con el consecuente encabezado de cada una de ellas en el mensaje, cosa que analizar y construir estos encabezados lleva tiempo. Si bien esto en las redes de área amplia no es significante, si lo es en una LAN.

Para el caso CLIENTE-SERVIDOR, se utiliza un protocolo solicitud-respuesta (request/reply), en vez del OSI (TCP/IP). El cliente envía un request pidiendo un servicio y el server lo recibe, realiza el trabajo y devuelve los datos pedidos o un código de error. Lo principal es su sencillez y su eficiencia.

Sencillez : No se tiene que establecer ninguna conexión sino hasta que esta se utilice, y el mensaje de respuesta sirve como agradecimiento a la solicitud.

Eficiencia : Las capas del protocolo son menos y por lo tanto más eficiente, si todas las máquinas fuesen idénticas, solo se necesitarían tres niveles: La Física, La de Enlace (ambas manejadas por Hardware), la de Solicitud/Respuesta (en lugar de la de sesión).

Las capas 3 y 4 no se utilizan pues no es necesario el ruteo ni tampoco se establecen conexiones. No existe administración de la sesión puesto que no existe y tampoco se utilizan las capas superiores.

Debido a que esta estructura es sencilla, se pueden reducir los servicios de comunicación que presta el micronúcleo a dos llamadas a sistema:

- SEND (dest, &mptr): envía el mensaje que apunta mptr al proceso destino y bloquea al proceso llamador hasta que se envíe el mensaje;
- RECEIVE (addr, &mptr): que hace que se bloquee el que realizó la llamada, hasta que llegue un mensaje, cuando llega este se copia en el buffer que apunta mptr y quien hizo la llamada se desbloquea. El parámetro addr, determina la dirección a la cual escucha el receptor.

21.2. - Direccionamiento [10][11][S.O.]

Hay varios métodos para el direccionamiento de los procesos:

1) *Integrar machine.number al código del proceso cliente* : machine indica el número de máquina dentro de la red y number, el número de proceso dentro de esa máquina. Pero este método no posee la transparencia que se busca ya que se está identificando que existen varias máquinas trabajando. A parte si se descompone esa máquina (server) se pierde el servicio pues los programas compilados tienen integrado ese número de máquina para ese servicio. Una variación de este esquema, utiliza machine.local_id.

2) *Dejar que los procesos elijan direcciones al azar y localizarlos mediante transmisiones* : En una LAN que soporte transmisiones, el emisor puede transmitir un paquete especial de localización con la dirección del proceso destino, todos los núcleos de las máquinas en la red reciben este mensaje y verifican si la dirección es la suya; en caso de que lo sea, regresa un mensaje "aquí estoy" con su dirección en la red (número de máquina). El núcleo emisor utiliza entonces esa dirección y la captura para uso posterior. Si bien esto cumple con las premisas, genera una carga adicional en el sistema.

3) *Generar un servidor de nombres* : Cada vez que se ejecute un cliente en su primer intento por utilizar un servidor, el cliente envía una solicitud al servidor de nombres (nombre en ASCII) para pedirle el número de la máquina donde se localiza el servidor. Una vez obtenida la dirección se puede enviar la solicitud de manera directa. El problema de este método es que es un componente centralizado y si bien se puede duplicar, presenta problemas en el mantenimiento de la consistencia.

Un método totalmente distinto, utiliza un hardware especial, dejando que los procesos elijan su dirección en forma aleatoria.

Sin embargo en vez de localizarlo mediante transmisiones en toda la red, los chips de interfase de la red se diseñan de modo que permitan a los procesos almacenar direcciones de procesos en ellos. Entonces las tramas usarían direcciones de procesos en vez de direcciones de máquinas. Al recibir cada trama, el chip de interfase de la red solo tendría que examinar la trama para saber si el proceso destino se encuentra en esa máquina. En caso afirmativo, la aceptaría, de lo contrario no.

21.3. - PRIMITIVAS BLOQUEANTES vs NO BLOQUEANTES [10][11][S.O.]

21.3.1. - SEND BLOQUEANTES (Primitivas sincrónicas) [10][11][S.O.]

Mientras que se envía el mensaje el proceso emisor se bloquea, de manera análoga el RECEIVE. En algunos casos el receptor puede especificar de quiénes quiere recibir el mensaje y queda bloqueado hasta que reciba el mensaje de él. La CPU está muerta, desperdiciando tiempo.

21.3.2. - SEND NO BLOQUEANTES (Primitivas asincrónicas) [10][11][S.O.]

Regresa de inmediato el control a quien hizo la llamada (send) antes de enviar el mensaje. La ventaja de esto es que puede trabajar en forma paralela con el envío del mensaje. Sin embargo existe la desventaja de no poder usar el buffer hasta que no se envíe la totalidad del mensaje. Una forma de solucionar esto es que el S.O. copie este buffer a una área propia y luego envíe el mensaje, liberando el buffer. Aquí se desperdicia tiempo en la copia.

21.3.3. - SEND SIN BLOQUEO CON INTERRUPCIÓN [10][11][S.O.]

El emisor es interrumpido cuando el mensaje fue enviado y el buffer está disponible. No se requiere de una copia, lo que ahorra tiempo, pero las interrupciones a nivel usuario dificultan mucho la programación. Maximiza el paralelismo.

En condiciones normales la primera opción es la mejor, no maximiza el paralelismo pero es fácil de comprender e implantar y no requiere el manejo del buffer en el núcleo. Generalmente el uso de SENDs bloqueantes o no bloqueantes se deja a los diseñadores, pues esto está muy ligado al problema que quieren solucionar. Aunque en algunos casos se dispone de los dos y el usuario elige su favorito.

21.4. - PRIMITIVAS ALMACENADAS EN BUFFER vs NO ALMACENADAS [10][11][S.O.]

Todas las primitivas anteriormente mencionadas son primitivas no almacenadas. Las primitivas almacenadas en buffer aparecen como una solución a una diferencia entre el Cliente y el Server en el envío y recepción de mensajes, o sea que si en las primitivas anteriormente mencionadas, el cliente envía un SEND antes de que el server ejecute un RECEIVE para esa dirección, se produce una pérdida del mensaje y con resultado de esto, el cliente reintentará la petición.

Si el cliente reintenta muchas veces puede ser que considere que el server se ha caído, o que la dirección no es correcta.

Entonces las primitivas almacenadas en buffer implican que el núcleo del receptor mantenga pendiente los mensajes por un instante usando un buffer lo que conlleva a un almacenamiento y manejo de mensajes que van llegando en forma prematura.

Una forma conceptualmente sencilla de enfrentar este manejo de los buffers es definir una nueva estructura llamada *buzón*.

Un proceso interesado en recibir mensajes le indica al núcleo que cree un buzón para él y especifica una dirección en la cual buscar los paquetes de la red. Así, todos los mensajes que lleguen a esa dirección se colocan en el buzón. El Receive elimina ahora un mensaje del buzón o se bloquea si no hay un mensaje presente.

En ciertos sistemas, no se deja que un proceso envíe un mensaje si no existe espacio para su almacenamiento en el destino. Para que este esquema funcione, el emisor debe bloquearse hasta que obtenga de regreso un reconocimiento, que indica que el mensaje ha sido recibido.

21.5. - PRIMITIVA CONFIABLES vs NO CONFIABLES [10][11][S.O.]

Confiable : Cuando se garantiza de alguna manera que el mensaje llega a destino.
No Confiable : Cuando no existe garantía alguna de que el mensaje hay sido entregado, podría haberse perdido.

Existen tres distintos enfoques de este problema:

1) Volver a definir la semántica SEND para hacerlo no confiable (el sistema no da garantía alguna acerca de la entrega de los mensajes).

2) Se envían reconocimientos de núcleo a núcleo (Server/Cliente) sin que estos se enteren. Así una solicitud de respuesta consta de cuatro mensajes: 1 solicitud, 2 reconocimiento, 3 respuesta, 4 reconocimiento.

3) El tercer método aprovecha la respuesta como método de reconocimiento. Si bien esto funciona bien, si la solicitud requiere de un cómputo extenso por parte del servidor, estaría mal descartar la respuesta antes de que el servidor estuviera seguro de que el cliente la haya recibido. Por esta razón a veces se utiliza un reconocimiento del núcleo del cliente al núcleo del servidor, permaneciendo éste bloqueado hasta ese entonces. Algunas veces al llegar la solicitud al servidor, si éste tarda mucho en resolverla, el servidor envía un reconocimiento antes de la respuesta.

Ejemplos:

	Con Conexión	**Sin Conexión**
Confiable	Transferencia de Arch. FTP	Correo Certificado (llega sino avisa)
No Confiable	Teléfono (puede haber Ruido)	Correo Simple (llega o no llega)

21.6. - IMPLEMENTACIÓN DEL MODELO CLIENTE-SERVIDOR [10][11][S.O.]

Existen muchas combinaciones posibles para la elección de un conjunto de primitivas de comunicación:

Elemento	**Opción 1**	**Opción 2**	**Opción 3**
Direccionamiento	Número de Máquina	Direcciones ralas de procesos	Búsqueda de nombres en ASCII por medio del Servidor
Bloqueo	Primitivas con bloqueo	Sin Bloqueo, copia al Núcleo	Sin Bloqueo con interrupciones
Almacenamiento en buffers	No usarlo descartar los mensajes inesperados	No usarlo mantenimiento temporario de mensajes inesperados	Buzones
Confiabilidad	No Confiable	Solicitud; Reconocimiento; Respuesta; Reconocimiento	Solicitud; Respuesta; Reconocimiento

21.7. - COMENTARIOS ACERCA DE IMPLEMENTACIÓN PROTOCOLOS Y SOFTWARE [10][11][S.O.]

Todas las redes tienen tamaño máximo de paquetes. Los mensajes mayores a esa cantidad, deben dividirse en varios paquetes y enviarse en forma independiente.

Uso de los reconocimientos, se pueden hacer por :

Paquete individual.

Ventaja : si se pierde un paquete, solo se retransmite el paquete;

Desventaja : se necesitan mas paquetes en la red,

o,

Reconocimiento por mensaje completo.

Ventaja : hay menos paquetes en la red;

Desventaja : Se debe retransmitir todo el mensaje en caso de pérdida.

En el modelo Cliente-Servidor, existen los siguientes tipos de mensajes que sirven para los siguiente casos:

Código	**Tipo de paquete**	**De**	**A**	**Descripción**
REQ	Solicitud	Cli	Ser	El cliente desea Servicio
REP	Respuesta	Ser	Cli	Respuesta del Servidor al Cliente
ACK	Reconocimiento	Cualq	Al Otro	El paquete anterior que ha llegado
AYA	Estas Vivo?	Cli	Ser	Verifica si el Servidor se ha descompuesto
IAA	Estoy Vivo	Ser	Cli	El Servidor no se ha descompuesto
TA	Intenta de Nuevo	Ser	Cli	El Servidor no tiene espacio
AU	Dirección desconocida	Ser	Cli	Ningún proceso utiliza esta dirección

Las secuencias más comunes son las siguientes:

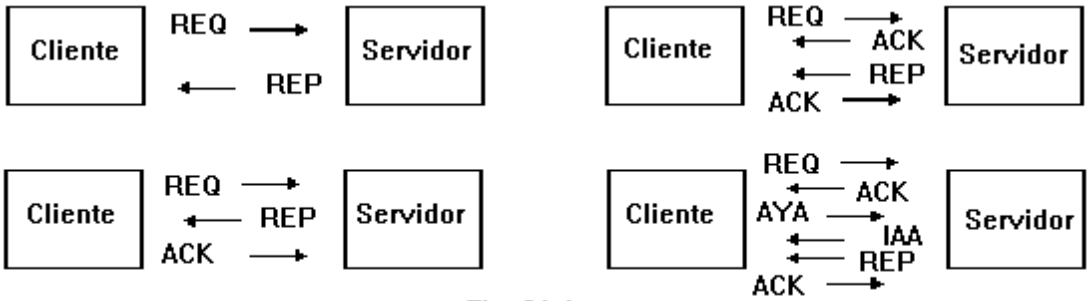

Fig. 21.1.