LISTA DE COTEJO REPORTE DE INVESTIGACION VALOR: 30 %

NOMBRE DEL DOCENTE: María de los Ángeles Pelayo Vaquero		
DATOS GENERALES DEL PROCESO DE EVALUACIÓN		
NOMBRE DEL ALUMNO: FONSECA ABRAJAN OSVANY JESUS		
PRODUCTO: INVESTIGACIÓN	UNIDAD: 2	PERIODO ESCOLAR: AGOSTO - DICIEMBRE 2024

INDICADOR	VALOR	PORCENTAJE OBTENIDO
Presentación - Formato	5 %	5 %
Introducción Idea clara del contenido del trabajo, motivando al lector a continuar con su lectura y revisión	5 %	5 %
Desarrolla el objetivo	2 %	2 %
Desarrollo de la investigación La investigación cumple con el tema solicitado	10 %	10 %
Desarrolla la conclusión de investigación	3 %	3 %
Gramática y ortografía	3 %	3 %
Bibliografía	2 %	2 %
Total	**30 %**	**30 %**

LISTA DE COTEJO ENSAYO VALOR: 30 %

NOMBRE DEL DOCENTE: María de los Ángeles Pelayo Vaquero		
DATOS GENERALES DEL PROCESO DE EVALUACIÓN		
NOMBRE DEL ALUMNO: FONSECA ABRAJAN OSVANY JESUS		
PRODUCTO: ENSAYO	UNIDAD: 2	PERIODO ESCOLAR: AGOSTO - DICIEMBRE 2024

INDICADOR	VALOR	PORCENTAJE OBTENIDO
Presentación - Formato	2 %	2 %
Introducción Idea clara del contenido del trabajo, motivando al lector a continuar con su lectura y revisión	5 %	5 %
Cuerpo del ensayo (empleo de artículos científicos)	10%	10%
Conclusión	5 %	5 %
Uso de formato de estilo	2 %	2 %
Gramática y ortografía	3 %	3 %
Bibliografía (APA o IEEE)	3 %	3 %
Total	**30 %**	**30 %**

ANEXOS

INSTITUTO TECNOLOGICO SUPERIOR DE SAN ANDRES TUXTLA

CARRERA:

ING.INFORMATICA

DOCENTE:

MARIA DE LOS ANGELES PELAYO VAQUERO

MATERIA:

SISTEMAS OPERATIVOS II

ACTIVIDAD:

INVESTIGACION

ALUMNO:

OSVANY JESUS FONSECA ABRAJAN

FECHA:

22/10/2024

NOMBRES, IDENTIFICADORES Y DIRECCIONES

Comencemos por dar un vistazo a lo que actualmente es un nombre. Un nombre dentro de un sistema distribuido es una cadena de bits o caracteres utilizados para hacer referencia a una entidad. En un sistema distribuido, una entidad puede ser prácticamente cualquier cosa. Ejemplos clásicos incluyen recursos tales como servidores, impresoras, discos y archivos. Otros ejemplos muy conocidos de entidades que a menudo se nombran de manera explícita son los procesos, usuarios, buzones de correo, grupos de noticias, páginas web, ventanas gráficas, mensajes, conexiones de red y muchas cosas más. Las entidades se pueden operar.

Por ejemplo, un recurso tal como una impresora ofrece una interfaz que contiene operaciones para imprimir un documento, solicitar el estado de una tarea de impresión y cosas similares. Además, una entidad tal como una conexión de red puede proporcionar operaciones para enviar y recibir datos, establecer parámetros de calidad de servicio, solicitar el estado y así sucesivamente. Para operar una entidad, es necesario acceder a ella, de modo que se requiere tener un punto de acceso. Un punto de acceso es otra clase, pero especial, de entidad en un sistema distribuido. Al nombre de un punto de acceso se le llama dirección. A la dirección del punto de acceso de una entidad se le denomina dirección de dicha entidad. Una entidad puede ofrecer más de un punto de acceso. Como comparación, un teléfono puede verse como un punto de acceso a una persona, en donde el número telefónico corresponde a una dirección. De hecho, en la actualidad mucha gente tiene distintos números de teléfono, y cada número corresponde a un punto en donde se le puede localizar. En un sistema distribuido, un ejemplo típico de un punto de acceso es un servidor que ejecuta un servidor específico, en el cual la dirección está formada por, digamos, una dirección IP y un número de puerto (es decir, la dirección del servidor a nivel de transporte). Una entidad puede modificar sus puntos de acceso en el curso del tiempo.

Por ejemplo, cuando una computadora móvil se mueve a una ubicación diferente, con frecuencia se le asigna una IP distinta a la que tenía antes. De manera similar, cuando una persona se traslada a otra ciudad u otro país, a menudo es necesario modificar también sus números telefónicos. De igual modo, cambiar de empleo o de proveedor de servicios de internet significa modificar nuestra dirección de correo electrónico. Entonces, una dirección es sólo una clase especial de nombre: hace referencia a un punto de acceso de una entidad. Debido a que un punto de acceso está íntimamente asociado con una entidad, parecería conveniente utilizar la dirección de un punto de acceso como nombre regular de la entidad asociada. Sin embargo, esto difícilmente se lleva a cabo dado que la nomenclatura es, por lo general, muy inflexible y a menudo nada amigable con el usuario.

Por ejemplo, no es poco común reorganizar un sistema distribuido con regularidad, de manera que un servidor específico pueda ejecutarse entonces en un servidor diferente al previo. La máquina anterior sobre la cual solía ejecutarse se puede reasignar a un servidor completamente diferente. En otras palabras, una entidad puede modificar fácilmente un punto de acceso, o un punto de acceso puede reasignarse a una entidad diferente. Si una dirección se utiliza para hacer referencia a una entidad, tendremos una referencia inválida en el instante en que el punto de acceso cambie o sea reasignado a otra entidad. Por tanto, es mucho mejor dejar que un servicio sea conocido por un nombre separado independientemente de la dirección del servidor asociado.

De modo similar, si una entidad ofrece más de un punto de acceso, no queda claro cuál dirección utilizar como referencia. Por ejemplo, muchas organizaciones distribuyen su servicio web a través de muchos servidores. Si utilizamos las direcciones asignadas a dichos servidores como referencia para un servicio web, no resulta evidente qué dirección se debe elegir como la mejor. De nuevo, una mucho mejor solución es la de tener un solo nombre para el servicio web independientemente de las direcciones de los diferentes servidores web.

Estos ejemplos ilustran que un nombre para una entidad que es independiente de sus direcciones con frecuencia es más fácil y flexible de usar. A dicho nombre se le conoce como independiente de su ubicación. Además de la dirección, existen otros tipos de nombres que merecen un trato especial, tales como los nombres empleados para identificar de manera única a una entidad

1. Un identificador hace referencia a una entidad como máximo.

2. Cada entidad es referida por al menos un identificador.

3. Un identificador siempre hace referencia a la misma entidad (es decir, nunca se reutiliza).

Mediante el uso de identificadores se vuelve más fácil hacer referencia clara a una entidad. Por ejemplo, asumamos que dos procesos hacen referencia a una entidad por medio de un identificador. Para verificar si los procesos hacen referencia a la misma entidad, es suficiente con verificar que los dos identificadores sean iguales. Dicha prueba no es suficiente si los dos procesos utilizan nombres regulares, no únicos, y no identificables. Por ejemplo, el nombre “Juan Pérez” no se puede tomar como una referencia única para una sola persona. De manera similar, si se puede reasignar una dirección a una entidad diferente, no podemos utilizar una dirección como un identificador. Consideremos el uso de números telefónicos, los cuales son razonablemente estables en el sentido de que un número telefónico hace referencia durante cierto tiempo a la misma persona o empresa. Sin embargo, usar un número telefónico como identificador no funcionará, ya que puede ser reasignado en el curso del tiempo. En consecuencia, la pastelería de Bob pudiera recibir llamadas telefónicas para la tienda de antigüedades de Alicia durante un largo periodo. En este caso, hubiera sido mejor utilizar un identificador verdadero para Alicia en lugar de su número telefónico.

Otro tipo importante de nombre es el destinado a utilizarse por las personas, también llamado nombre amigable para el usuario. Al contrario de las direcciones y los identificadores, un nombre amigable para el usuario por lo general se representa como una cadena de caracteres. Estos nombres aparecen de muchas formas diferentes. Por ejemplo, en los sistemas UNIX los archivos tienen cadenas de caracteres como nombres que pueden tener una longitud de hasta 255 caracteres, y los cuales son definidos completamente por el usuario. De manera similar, los nombres DNS están representados como sencillas cadenas de caracteres sensibles a mayúsculas y minúsculas.

Tener nombres, identificadores, y direcciones nos da el tema central del presente capítulo, ¿cómo resolver nombres e identificadores de direcciones? Antes de abordar distintas soluciones, es importante darnos cuenta de que existe a menudo una relación cercana entre la resolución de nombres en los sistemas distribuidos y el ruteo de mensajes. En principio, un sistema de nombres mantiene un vínculo nombre a dirección el cual es, en su forma más simple, solamente una tabla (nombre, dirección) de pares. Sin embargo, en sistemas distribuidos que se expanden en grandes redes y para las cuales se requieren muchos recursos, una tabla centralizada no funcionará.

Referencia bibliográfica

(S/f). Google.com. Recuperado el 22 de octubre de 2024, de https://drive.google.com/file/d/1NrmIiOY_zErWP92AJYbeMGn9v4_XYN38/view

INSTITUTO TECNOLOGICO SUPERIOR DE SAN ANDRES TUXTLA

CARRERA:

ING.INFORMATICA

DOCENTE:

MARIA DE LOS ANGELES PELAYO VAQUERO

MATERIA:

SISTEMAS OPERATIVOS II

ACTIVIDAD:

ENSAYO

ALUMNO:

OSVANY JESUS FONSECA ABRAJAN

FECHA:

03/10/2024

Introducción

En este capítulo, damos un vistazo de cerca a la forma en que los distintos tipos de procesos cumplen su crucial papel en los sistemas distribuidos. El concepto de proceso tiene su origen en el campo de los sistemas operativos donde, por lo general, se define como un programa en ejecución. Desde la perspectiva de un sistema operativo, la administración y la calendarización de los procesos son quizá los asuntos más importantes por tratar; sin embargo, cuando nos referimos a los sistemas distribuidos, tenemos que otros temas resultan ser igualmente o más importantes. Por ejemplo, para organizar los sistemas cliente-servidor de manera eficiente, por lo general conviene más utilizar técnicas multihilos. Tal como explicamos en la primera sección, una de las contribuciones más importantes de los hilos a los sistemas distribuidos es que permiten la construcción de clientes y servidores de un modo en el que la comunicación y el procesamiento local se pueden traslapar, ello origina un alto nivel de rendimiento. En los años más recientes, el concepto de virtualización ha ganado popularidad. La virtualización permite a una aplicación, e incluso a un ambiente completo incluyendo el sistema operativo, ejecutarse en forma concurrente con otras aplicaciones, pero altamente independiente de su hardware y plataforma subyacentes, provocando un alto grado de portabilidad. Más aún, la virtualización permite aislar las fallas ocasionadas por errores o problemas de seguridad

Hilos

Aunque los procesos forman un bloque de construcción en los sistemas distribuidos, la práctica indica que la granularidad de los procesos, como tal es proporcionada por los sistemas operativos en los cuales los sistemas distribuidos son construidos, pero esto no es suficiente. En lugar de ello, resulta que tener un mayor grado de granularidad en la forma de múltiples hilos de control por proceso vuelve mucho más fácil construir aplicaciones distribuidas y obtener un mejor rendimiento. En esta sección daremos un vistazo cercano al rol que cumplen los hilos en los sistemas distribuidos y explicaremos por qué son tan importantes.

Introducción a los hilos

Para entender el rol de los hilos en los sistemas distribuidos, es importante comprender lo que es un proceso y cómo se relacionan los procesos y los hilos. Para ejecutar un programa, un sistema operativo crea cierto número virtual de procesadores, cada procesador ejecuta un programa diferente. Con el fin de seguir la pista de estos procesadores virtuales, el sistema operativo tiene una tabla de procesos que contiene entradas para almacenar valores de los registros de la CPU, mapas de memoria, archivos abiertos, información contable, privilegios, etc. Con frecuencia, un proceso está definido como un programa en ejecución, esto es, un programa que es ejecutado por lo general en uno de los procesadores virtuales del sistema operativo. Una cuestión importante es que el sistema operativo tiene mucho cuidado en asegurar que los procesos independientes no puedan afectar de manera maliciosa o inadvertida la corrección del comportamiento de otro proceso. En otras palabras, garantizar que muchos procesos puedan compartir de manera concurrente la misma CPU, así como otros recursos de hardware, en forma transparente. Por lo general, el sistema operativo requiere soporte de hardware para reforzar esta separación.

Uso de hilos en sistemas no distribuidos

Antes de explicar el rol de los hilos en los sistemas distribuidos, consideremos su uso en sistemas tradicionales no distribuidos. Existen distintos beneficios de los procesos multihilos que han incrementado la popularidad hacia el uso de sistemas mediante hilos. El principal beneficio proviene de que, en un proceso con un único hilo de control, siempre que se ejecuta una llamada de sistema se bloquea el proceso como un todo. Para ejemplificar, consideremos una aplicación tal como una hoja de cálculo, y asumamos que un usuario desea modificar los valores de manera continua e interactiva. Una propiedad importante de un programa de hoja de cálculo es que mantiene las dependencias funcionales entre las diferentes llamadas, con frecuencia desde diferentes hojas de cálculo.

Otra ventaja de la tecnología multihilos es que hace posible explotar el paralelismo cuando ejecutamos el programa dentro de un sistema multiprocesador. En ese caso, a cada hilo se le asigna una CPU diferente mientras los datos compartidos se almacenan en una memoria principal compartida. Cuando lo diseñamos de manera apropiada, dicho paralelismo puede ser transparente: el proceso se ejecutará igual de bien en un sistema uniproceso, pero un poco más lento. La tecnología multiproceso para el paralelismo ha adquirido cada vez más importancia con la disponibilidad de estaciones de trabajo multiproceso relativamente baratas. Tales sistemas de cómputo son usados típicamente en la ejecución de servidores y de aplicaciones cliente-servidor.

La tecnología multihilos también resulta muy útil en el contexto de las grandes aplicaciones. Con frecuencia tales aplicaciones son desarrolladas como una colección de programas cooperativos para que cada programa sea ejecutado mediante un proceso aparte. Este método es típico de un ambiente UNIX. La cooperación entre programas se implementa por medio de mecanismos de comunicación interproceso (IPC, por sus siglas en inglés). Para sistemas UNIX, estos mecanismos incluyen generalmente tuberías (pipes), colas de mensajes, y segmentos de memoria compartida [vea también Stevens y Rago (2005)]. La mayor desventaja de todos los mecanismos IPC es que a menudo la comunicación requiere de un excesivo intercambio de contexto, como lo ilustra la figura 3-1.

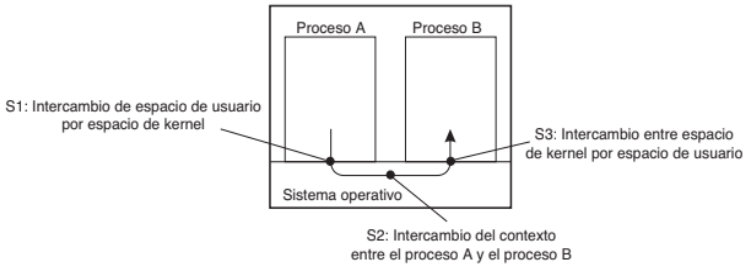

Figura 3-1. Intercambio de contexto como resultado de IPC.

Implementación de hilos

Por lo general, los hilos son proporcionados en la forma de un paquete de hilos. Dicho paquete contiene operaciones para crear y destruir hilos, así como operaciones con respecto a la sincronización de variables tales como los mútex y las variables de condición. Existen básicamente dos métodos para implementar un paquete de hilos. El primer método es la construcción de una biblioteca de hilos que se ejecutan por completo en el modo de usuario; el segundo es que el kernel esté al tanto de los hilos y los pueda calendarizar.

Una biblioteca de hilos a nivel de usuario tiene algunas ventajas. Primero, es barato crearla y destruirla. Debido a que toda la administración de los hilos se mantiene en el espacio de direcciones del usuario, el precio de la creación de un hilo es determinado primordialmente por el costo de la ubicación de memoria para establecer una pila de hilos. De manera análoga, destruir un hilo involucra liberar la memoria para la pila, la cual ya no es utilizada. Las dos operaciones son baratas.

Otra ventaja de los hilos a nivel de usuario es que con frecuencia el intercambio de un contexto de hilo puede realizarse mediante unas cuantas instrucciones. Básicamente, sólo se requiere almacenar los valores de los registros de la CPU y, posteriormente, recargarla con valores almacenados previamente del hilo al cual se hace el intercambio. No hay necesidad de modificar los mapas de memoria, el reinicio del TLB, ni hacer un conteo de la CPU, etc. El intercambio de contextos de hilo se efectúa cuando dos hilos requieren sincronización, por ejemplo, cuando entramos a una sección de datos compartidos.

Sin embargo, una desventaja importante de los hilos a nivel de usuario es que, al invocar una llamada de bloqueo de sistema, ésta bloqueará todo el proceso al cual pertenece el hilo, y entonces bloqueará todos los hilos presentes en dicho proceso. Como ya explicamos, los hilos son particularmente útiles para estructurar grandes aplicaciones como partes que podemos ejecutar de manera lógica al mismo tiempo. En tal caso, el bloqueo de E/S pudiera no prevenir a otras partes de ser ejecutadas en la máquina. Para tales aplicaciones, los hilos a nivel usuario no ayudan.

Una solución radica en una forma híbrida entre hilos de nivel usuario y nivel kernel, por lo general se le conoce como procesos de peso ligero (LWP, por sus siglas en inglés). Un LWP se ejecuta en el contexto de un solo proceso (de peso completo), y puede haber distintos LWP por proceso. Además de contar con LWP, un sistema ofrece además un paquete de hilos a nivel usuario, lo cual permite a las aplicaciones efectuar las operaciones necesarias para crear y destruir hilos. Además, el paquete proporciona los medios para la sincronización de hilos, tales como el mútex y variables de condición. El asunto importante es que el paquete de hilos está implementado por completo en el espacio de usuario. En otras palabras, todas las operaciones sobre hilos son procesadas sin la intervención de un kernel.

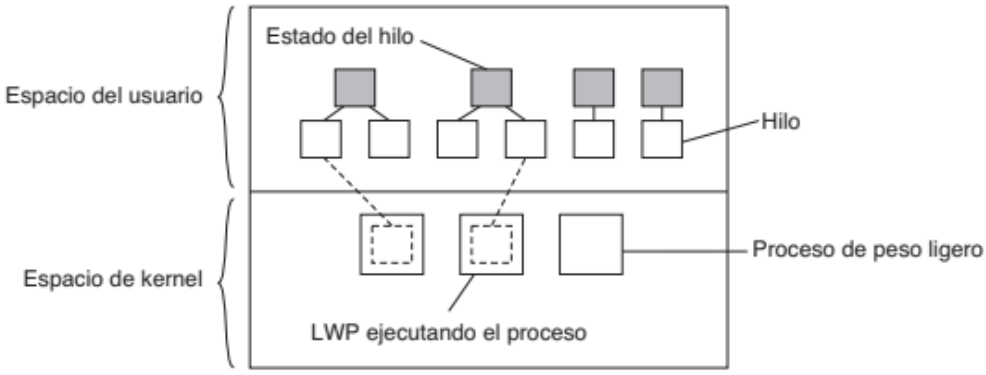

Figura 3-2. Combinación de procesos a nivel de kernel de peso ligero e hilos de nivel usuario.

La combinación de (a nivel usuario) hilos y LWP trabaja de la siguiente manera. El paquete de hilos tiene una rutina simple para calendarizar el siguiente hilo. Al crear un LWP (lo cual se hace por medio de una llamada de sistema), el LWP cuenta con su propia pila, y tiene la instrucción de ejecutar la rutina de calendarización en busca de un hilo para ejecución. Si existen varios LWP, entonces cada uno invoca al planificador. La tabla de hilos, utilizada para seguir la pista del conjunto de hilos actuales, es compartida por los LWP. Al proteger esta tabla garantizamos que el acceso mutuamente exclusivo se lleve a cabo por medio de los mútex que se implementan por completo en el espacio de usuario. En otras palabras, la sincronización entre LWP no requiere soporte alguno del kernel.

Hilos en sistemas distribuidos

Una propiedad importante de los hilos es que pueden proporcionar un medio conveniente para permitir llamadas de bloqueo de sistema sin bloquear todo el proceso en que se ejecuta el hilo. Esta propiedad vuelve a los hilos particularmente atractivos para su uso dentro de sistemas distribuidos ya que es mucho más fácil expresar la comunicación mediante múltiples conexiones lógicas al mismo tiempo.

Clientes multihilos

Para establecer un alto grado de transparencia de distribución, los sistemas distribuidos que operan en redes de área amplia pudieran necesitar la conciliación de grandes tiempos de propagación de mensajes de interproceso. En redes de área amplia, los ciclos tienen retrasos que pueden rondar fácilmente el orden de cientos de milisegundos, o incluso segundos en algunas ocasiones. La manera más común de ocultar las latencias de comunicación es iniciar la comunicación y proceder de inmediato con alguna otra cosa. Un ejemplo común en donde sucede esto es en los navegadores web. En muchos casos, un documento web consta de un archivo HTML que contiene texto plano junto con una colección de imágenes, iconos, etc. Para traer cada elemento de un documento web, el navegador tiene que configurar una conexión TCP/IP, leer los datos de entrada, y pasarlos hacia el componente de visualización. Configurar la conexión, así como leer los datos de entrada que son inherentes a las operaciones de bloqueo. Al tratar con el transporte para la comunicación de grandes volúmenes, también tenemos la desventaja de que el tiempo necesario para completar cada operación podría ser relativamente largo.

Servidores multihilos

Aunque existen importantes beneficios para los clientes multihilos, como hemos visto, en los sistemas distribuidos el principal uso de la tecnología multihilos está del lado del servidor. La práctica muestra que la tecnología multihilos no solamente simplifica el código del servidor de manera considerable, sino que además hace más sencillo el desarrollo de servidores que explotan el paralelismo para lograr un alto rendimiento, incluso en sistemas de un solo procesador. Sin embargo, ahora que las computadoras multiproceso están ampliamente disponibles como estaciones de trabajo de propósito general, aplicar la tecnología multihilos para implementar el paralelismo es aún más útil.

Para comprender los beneficios de los hilos para escribir código del servidor, consideremos la organización de un servidor de archivos que ocasionalmente se tiene que bloquear en espera del disco. Por lo general, el servidor de archivos espera una petición de entrada para una operación de archivo, posteriormente ejecuta la petición, y luego envía la respuesta de regreso. En la figura 3-3 podemos ver una posible y popular organización en particular. Aquí, un hilo servidor, lee las peticiones de entrada para una operación con archivos. Las peticiones son enviadas por clientes hacia un punto final bien conocido por este servidor. Después de examinar la petición, el hilo servidor elije un hilo trabajador sin utilizar (es decir, bloqueado) y le agrega la petición.

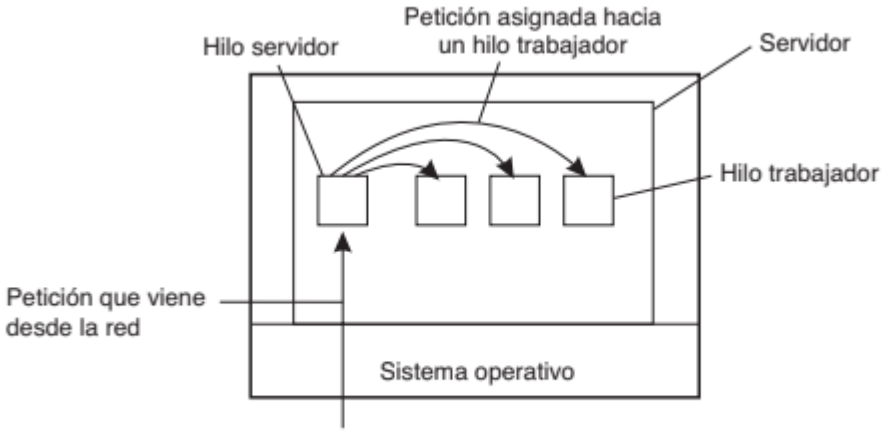

Figura 3-3. Servidor multihilos organizado en un modelo servidor/trabajador.

El trabajador procede a realizar una lectura de bloqueo en el sistema de archivos local, ello puede provocar que el hilo se suspenda hasta que los datos sean recuperados desde el disco. Si el hilo se suspende, se selecciona otro hilo para su ejecución. Por ejemplo, el hilo servidor puede seleccionar la adquisición de más trabajo. De manera alternativa, se puede seleccionar otro hilo trabajador que esté listo para su ejecución.

Virtualización

Podemos ver los hilos y los procesos como una manera de hacer más cosas al mismo tiempo. En efecto, nos permiten construir (partes de) programas que parecen ejecutarse en forma simultánea. En una computadora con un solo procesador, esta ejecución simultánea es, por supuesto, una ilusión. Dado que solamente contamos con una CPU, sólo se ejecuta una instrucción de un proceso o hilo a la vez. El rápido intercambio entre hilos y procesos crea la ilusión de paralelismo. La separación entre tener una sola CPU y ser capaz de pretender que existen más unidades de procesamiento se puede extender también a otros recursos, ello origina lo que conocemos como virtualización de recursos. Esta virtualización se ha aplicado durante muchas décadas, pero ha adquirido un renovado interés conforme los sistemas computacionales (distribuidos) se han vuelto más populares y complejos, originando una situación en la cual el software de aplicación por lo general sobrevive a sus sistemas de hardware y software subyacentes.

El rol de la virtualización en los sistemas distribuidos

En la práctica, cada sistema de cómputo (distribuido) ofrece una interfaz de programación hacia un nivel de software más alto, como lo muestra la figura 3-5(a). Existen muchos tipos diferentes de interfaces, abarcando desde un conjunto básico de instrucciones —como el ofrecido por una CPU—hasta una vasta colección de interfaces de programación de aplicaciones que vienen con muchos de los sistemas de middleware actuales. En su esencia, la virtualización trata con la extensión o el reemplazo de una interfaz existente de modo que imite el comportamiento de otro sistema, como vemos en la figura 3-5(b). Pronto llegaremos a la explicación de los detalles técnicos de la virtualización, pero primero nos concentraremos en el porqué es importante la virtualización para los sistemas distribuidos.

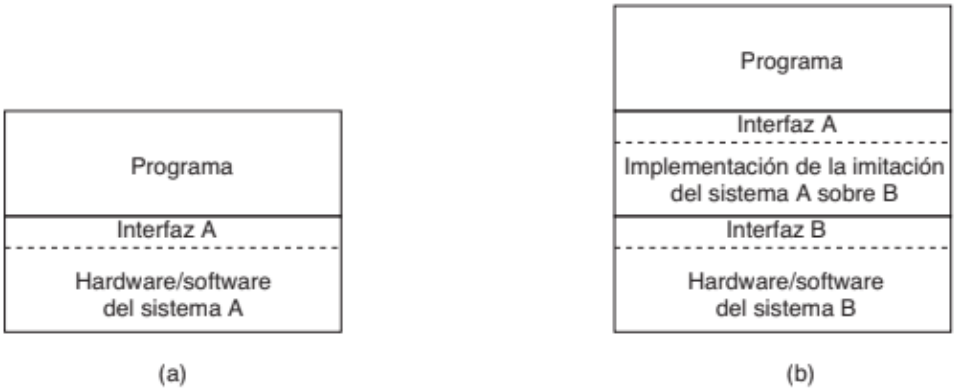

Figura 3-5. (a) Organización general entre un programa, la interfaz y el sistema. (b) Organización general entre el sistema de virtualización A sobre el sistema B.

Arquitecturas de máquinas virtuales

Para comprender las diferencias en la virtualización, es importante darse cuenta de que, por lo general, los sistemas de cómputo ofrecen cuatro tipos de interfaz distintos, y en cuatro niveles diferentes:

1. Una interfaz entre el hardware y el software, constituida por instrucciones máquina que se pueden invocar desde cualquier programa.

2. Una interfaz entre el hardware y el software, constituida por instrucciones máquina que se pueden invocar solamente desde programas privilegiados, tales como los sistemas operativos.

3. Una interfaz que consta de llamadas de sistema como las que ofrece un sistema operativo.

4. Una interfaz que consta de llamadas a bibliotecas, las cuales forman, por lo general, lo que conocemos como interfaz de programación de aplicaciones (API, por sus siglas en inglés). En muchos casos, las llamadas de sistema ya mencionadas están ocultas por una API.

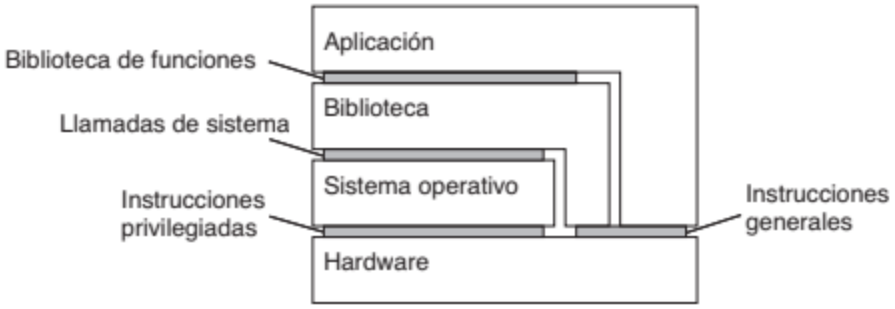

Figura 3-6. Distintas interfaces ofrecidas por los sistemas de cómputo.

Un método alternativo hacia la virtualización es el que proporciona un sistema que básicamente se implementa como una capa que cubre por completo al hardware original, pero que ofrece todo un conjunto de instrucciones del mismo (o de otro hardware) como una interfaz. Resulta crucial el hecho de que se puede ofrecer esta interfaz de manera simultánea a diferentes programas. Como resultado, ahora es posible tener múltiples y diferentes sistemas operativos que se ejecutan de distinto modo y concurrentemente sobre la misma plataforma. A esta capa, por lo general, la conocemos como el monitor de la máquina virtual (VMM, por sus siglas en inglés). Ejemplos típicos de este enfoque son VMware (Sugerman y cols., 2001) y Xen (Barham y cols., 2003). En la figura 3-7 se muestran estos dos métodos.

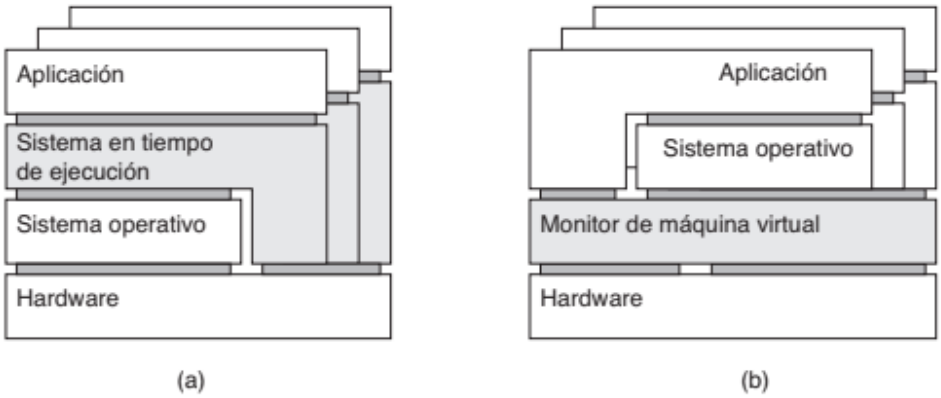

Figura 3-7. (a) Un proceso en una máquina virtual, con múltiples instancias de (aplicación, tiempo de ejecución) combinaciones. (b) Un monitoreo en una máquina virtual, con múltiples instancias de (aplicaciones, sistema operativo) combinaciones.

Tal como argumentan Rosenblum y Garfinkel (2005), las VMM serán cada vez más importantes en el contexto de la confiabilidad y la seguridad para los sistemas (distribuidos). Dado que permiten el aislamiento de toda una aplicación y de su ambiente, una falla ocasionada por un error o un ataque a la seguridad no afectará a una máquina en su totalidad. Además, como ya mencionamos, la portabilidad mejora de manera importante dado que las VMM proporcionan un desacoplamiento posterior entre hardware y software, lo cual permite mover un ambiente completo desde una máquina a otra.

Clientes

En los capítulos anteriores explicamos el modelo cliente-servidor, los roles de los clientes y de los servidores, y las maneras en que interactúan. Ahora veamos de cerca la anatomía de clientes y servidores, respectivamente. Comenzamos esta sección con una explicación relacionada con los clientes. En la siguiente sección explicaremos los servidores.

Interfaces de usuario en red

Una tarea importante de las máquinas cliente es la de proporcionar los medios necesarios para que los usuarios interactúen con servidores remotos. Existen básicamente dos maneras soportadas para efectuar esta interacción. Primero, para cada servicio remoto, la máquina cliente tendrá una contraparte por separado que puede contactar el servicio sobre una red. Un ejemplo clásico es una agenda ejecutando en la PDA de un usuario que requiere sincronizarse con una remota, posiblemente una agenda compartida. En este caso, un protocolo a nivel de aplicación manipulará esta sincronización, como podemos ver en la figura 3-8(a).

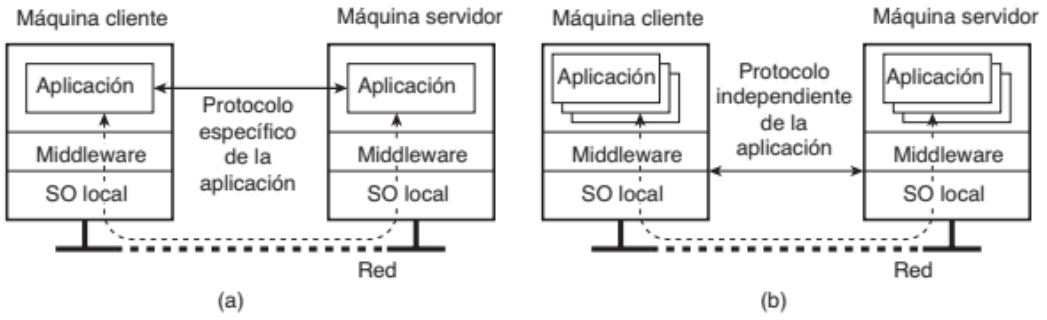

Figura 3-8. (a) Aplicación en red con su propio protocolo. (b) Solución general que permite el acceso a aplicaciones remotas.

Una segunda solución es proporcionar acceso directo a servicios remotos solamente con la oferta de una interfaz de usuario adecuada. En efecto, esto significa que la máquina cliente sólo se utiliza como terminal sin necesidad de almacenamiento local, ello produce una solución neutral para la aplicación tal como vemos en la figura 3-8(b). En el caso de interfaces de usuario en red, todo es procesado y almacenado en el servidor. Este método de cliente delgado recibe mayor atención al incrementarse la conectividad a internet, y los dispositivos hand-held se han vuelto cada vez más sofisticados. Como argumentamos en el capítulo anterior, las soluciones de cliente delgado también son populares ya que facilitan las tareas de administración de sistemas. Demos un vistazo a la manera en que se da soporte a las interfaces de usuario en red.

Clientes delgados en redes de cómputo

Por evidentes razones, las aplicaciones manipulan una pantalla mediante el uso de comandos específicos de pantalla proporcionados por X. Por lo general, estos comandos se envían a través de la red en donde posteriormente el kernel X los ejecuta. Debido a su naturaleza, las aplicaciones escritas para X deben preferentemente separar la lógica de la aplicación a partir de los comandos de la interfaz de usuario.

Tanto del lado del envío como del de la recepción se mantiene un caché local para buscar las entradas mediante el uso del identificador del mensaje. Cuando se envía un mensaje, busca primero en el caché local. Si lo encuentra, esto significa que se envió un mensaje previo con el mismo identificador, pero posiblemente con diferentes datos. En ese caso, se utiliza una codificación diferencial para enviar solamente las diferencias entre los dos. Del lado de la recepción, el mensaje busca también en el caché local, después de lo cual puede emprender la decodificación de las diferencias. En el uso del caché se utilizan técnicas de compresión estándar, las cuales por lo general provocan un aumento del cuádruple del rendimiento en el ancho de banda. Sobre todo, esta técnica a reportado reducción del ancho de banda hasta en un factor de 1000, lo cual permite a X ejecutar también enlaces de bajo ancho de banda de solamente 9 600 kbps.

En THINC, las peticiones de despliegue desde la aplicación son interceptadas y traducidas dentro de comandos de más bajo nivel. Mediante la intercepción de peticiones de la aplicación, THINC puede hacer uso de la semántica de la aplicación para decidir cuál combinación de bajo nivel debemos utilizar. Los comandos traducidos no se envían de inmediato a la pantalla, en lugar de eso se forman en una fila. Al formar en un proceso por lotes distintos comandos es posible agregar comandos de pantalla dentro de una fila, ello disminuye el número de mensajes. Por ejemplo, cuando un nuevo comando para dibujar una región particular de la pantalla sobrescribe de manera efectiva lo que pudo haber establecido un comando previo (y aún en formación), este último debe ser enviado a la pantalla. Finalmente, en lugar de dejar que la pantalla solicite refrescarse, THINC siempre empuja las actualizaciones al volverse disponible. Este método de empujar ahorra latencia ya que no requiere el envío de una petición de actualización por parte de la pantalla.

Software del lado del cliente para transparencia de la distribución

El software del cliente está compuesto por más que solamente interfaces de usuario. En muchos casos, en una aplicación cliente-servidor las partes del proceso y el nivel de datos también se ejecutan del lado del cliente. Una clase especial está formada por software embebido del cliente, digamos como el de cajeros bancarios automáticos (ATM, por sus siglas en inglés), cajas registradoras, lectores de código de barras, cajas TV set-top, etc. En estos casos, la interfaz de usuario es una parte relativamente pequeña del software de la cliente comparada con el procesamiento local y los medios de comunicación. Además de la interfaz de usuario y de otro software relacionado con la aplicación, el software del cliente comprende los componentes necesarios para lograr la transparencia de distribución. De manera ideal, un cliente no debiera estar al tanto de que se está comunicando con procesos remotos. Por contraste, la transparencia de distribución es con frecuencia menos transparente para los servidores por razones de rendimiento y de precisión. Por ejemplo, en el capítulo 6 mostraremos que los servidores replicados en ocasiones requieren comunicarse para establecer la manera y el orden en que deben realizarse las operaciones en cada réplica. De manera similar, muchos sistemas distribuidos implementan la transparencia de replicación por medio de soluciones del lado del cliente. Por ejemplo, imagine un sistema distribuido con servidores replicados. Tal replicación se puede lograr mediante el reenvío de una petición a cada réplica, como aparece en la figura 3-10. El software del lado del cliente puede recopilar de manera transparente todas las respuestas y pasar solamente una respuesta a la aplicación del cliente.

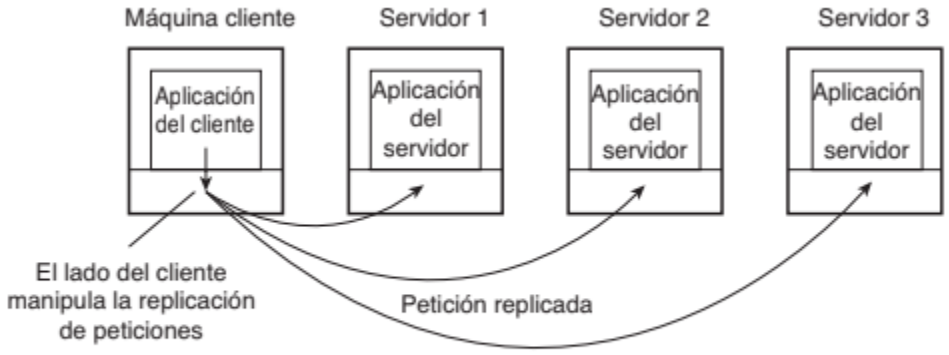

Figura 3-10. Transparencia de replicación de un servidor mediante una solución del lado del cliente.

Servidores

Ahora demos un vistazo a la organización de los servidores. En las páginas siguientes, nos concentraremos primero en diversos asuntos relacionados con el diseño general de los servidores, para continuar con la explicación de clústeres de servidores.

Temas generales de diseño

Un servidor es un proceso que implementa un servicio específico en representación de un conjunto de clientes. En esencia, cada servidor está organizado en la misma forma: espera una petición entrante de un cliente y se asegura de que se haga algo con dicha petición, después de lo cual espera por la siguiente petición entrante. Existen distintas maneras de organizar servidores. En el caso de un servidor iterativo, el propio servidor manipula la petición y, si es necesario, devuelve una respuesta a la petición del cliente. Un servidor concurrente no manipula por sí mismo la petición, pero la pasa a un hilo separado o a otro proceso, después de lo cual de inmediato queda en espera de la siguiente petición entrante. Un servidor multihilos es ejemplo de un servidor concurrente. Una implementación alternativa para un servidor concurrente es la de crear un nuevo proceso para cada nueva petición entrante. Este método es seguido en muchos sistemas UNIX. El hilo del proceso que manipula la petición es responsable de devolver una respuesta a la petición del cliente.

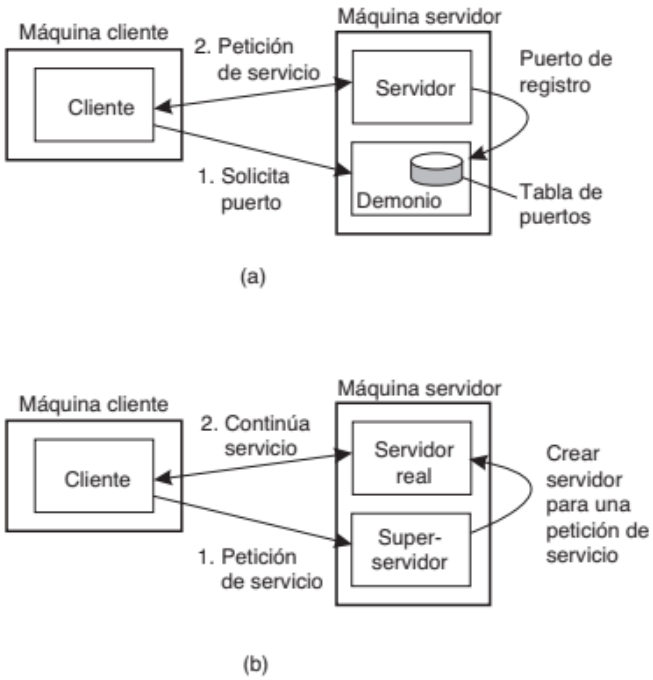

Figura 3-11. (a) Enlace cliente-servidor mediante el uso de un demonio. (b) Enlace cliente-servidor mediante el uso de un superservidor.

Servidores de clústeres

En el capítulo 1 explicamos brevemente la computación en clúster como una de las muchas apariencias de los sistemas distribuidos. Ahora daremos un vistazo a la organización de los servidores de clústeres, junto con los problemas que surgen en su diseño.

Organización general

Visto de manera simple, un servidor de clúster no es otra cosa que una colección de máquinas conectadas a través de una red, donde cada máquina ejecuta uno o más servidores. Los servidores de clústeres que consideramos aquí son aquellos en los cuales las máquinas están conectadas mediante una red de área local, con frecuencia ofreciendo un gran ancho de banda y latencia muy pequeña.

El primer nivel consta de un interruptor (switch) lógico a través del cual se rutean las peticiones del cliente. Tal interruptor puede variar ampliamente. Por ejemplo, los interruptores ubicados en la capa de transporte aceptan peticiones de conexión TCP entrantes y pasan dichas peticiones a uno de los servidores incluidos en el clúster, como explicaremos más adelante. Un ejemplo completamente diferente es un servidor web que acepta peticiones HTTP entrantes, pero que permite el paso parcial de peticiones a los servidores de aplicación para un proceso posterior solamente para después recopilar los resultados y devolver una respuesta HTTP.

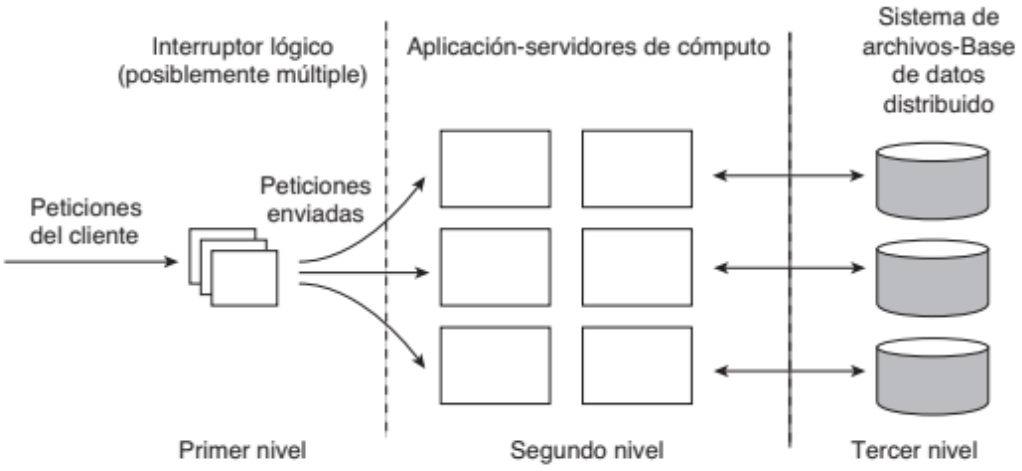

Figura 3-12. Organización general del cluster de servidores de tres niveles.

Servidores distribuidos

Los servidores de clústeres explicados hasta este momento son configurados generalmente de manera estática. En estos clústeres, con frecuencia existe una máquina separada para efectuar la administración que mantiene la huella de los servidores disponibles, y pasa esta información a otras máquinas conforme sea apropiado, tales como el switch. Como ya mencionamos, la mayoría de los clústeres de información ofrecen un solo punto de acceso. Cuando este punto falla, el clúster se vuelve no disponible. Para eliminar este potencial problema podemos proporcionar varios puntos de acceso, de los cuales se hacen públicas las direcciones. Por ejemplo, el Servicio de Nombres de Dominio (DNS, por sus siglas en inglés) puede devolver diversas direcciones, todas pertenecientes al mismo nombre del servidor. Este método también requiere que los clientes realicen varios intentos si uno de los servidores falla. Más aún, esto no resuelve el problema de requerimiento de puntos de acceso estáticos.

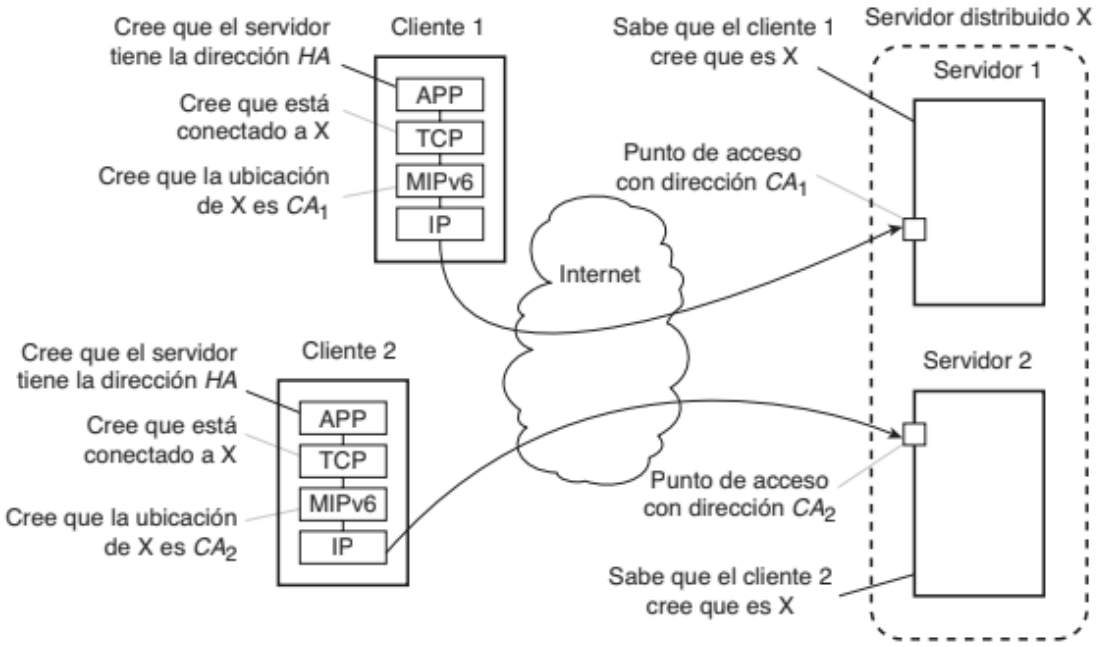

Figura 3-14. Optimización de ruta en un servidor distribuido.

Administración de servidores de clústeres

Un servidor de clúster debe aparecer ante el mundo exterior como una sola computadora, como en realidad es frecuentemente el caso. Sin embargo, cuando se trata de dar mantenimiento al clúster, la situación cambia de manera dramática. Se han hecho diversos intentos para facilitar la administración de un servidor de clúster, según veremos a continuación.

Métodos comunes

Por mucho, el método más común para administrar un servidor de clúster es extender las funciones tradicionales de administración de una sola computadora al clúster. En su modo más primitivo, esto significa que un administrador puede acceder a un nodo desde un cliente remoto y ejecutar comandos locales de administración para monitorear, instalar, y modificar los componentes. Algo más avanzado es ocultar el hecho de que se necesita acceder a un nodo y en lugar de eso proporcionar una interfaz a una máquina para la administración que permita recopilar la información desde uno o más servidores, actualizar sus componentes, agregar y remover nodos, etc. La ventaja principal del último método es que las operaciones colectivas, que operan en un grupo de servidores, pueden proporcionarse de manera más fácil. Este tipo de administración de servidores de clústeres se aplica extensamente en la práctica, y se puede ejemplificar mediante un software de administración como Clúster Systems Management de IBM (Hochstetler y Beringer, 2004).

Conclusión

Todos estos conceptos están interrelacionados en el mundo de la computación moderna. Los procesos e hilos son fundamentales para el rendimiento y la ejecución eficiente de aplicaciones dentro de sistemas operativos y entornos virtualizados. La virtualización permite que múltiples sistemas operativos y aplicaciones se ejecuten en un único servidor físico, optimizando el uso de los recursos. En un entorno cliente-servidor, los servidores, virtualizados o no, responden a las solicitudes de múltiples clientes simultáneamente, lo que requiere la capacidad de manejar múltiples hilos y procesos para garantizar un servicio eficiente y escalable. Este conjunto de tecnologías y conceptos permite que los sistemas informáticos sean más flexibles, eficientes y capaces de manejar cargas de trabajo complejas, especialmente en la era de la computación en la nube y las arquitecturas distribuidas.

INSTITUTO TECNOLOGICO SUPERIOR DE SAN ANDRES TUXTLA

CARRERA:

ING.INFORMATICA

DOCENTE:

MARIA DE LOS ANGELES PELAYO VAQUERO

MATERIA:

SISTEMAS OPERATIVOS II

ACTIVIDAD:

ENSAYO

ALUMNO:

OSVANY JESUS FONSECA ABRAJAN

FECHA:

04/10/2024

Introducción

La comunicación entre procesos se encuentra en el núcleo de todos los sistemas distribuidos. No tiene sentido estudiar los sistemas distribuidos, sin examinar cuidadosamente las formas en que los procesos desarrollados en diferentes máquinas pueden intercambiar información. En los sistemas distribuidos, la comunicación siempre se basa en el paso de mensajes de bajo nivel, tal como lo ofrece la red subyacente. Expresar la comunicación a través del paso de mensajes, es más difícil que utilizar primitivas basadas en memoria compartida, como se hace en plataformas no distribuidas. Los sistemas distribuidos modernos, con frecuencia consisten en miles o incluso millones de procesos esparcidos en una red con comunicación poco confiable como internet. A menos que las primitivas de comunicación para redes de computadoras sean remplazadas por algo más, el desarrollo de aplicaciones de gran escala es extremadamente difícil. El objetivo de este ensayo es analizar los mecanismos de comunicación en los sistemas operativos, explorando cómo estos sistemas facilitan la interacción entre los procesos y dispositivos. Se examinarán los distintos métodos de comunicación entre procesos (IPC), como señales, semáforos, y memoria compartida, así como las formas en que los sistemas operativos gestionan y optimizan el intercambio de datos y la coordinación entre aplicaciones y hardware. Además, se buscará comprender cómo estos mecanismos contribuyen al rendimiento y estabilidad del sistema en su conjunto.

Protocolos en capas

Debido a la ausencia de memoria compartida en los sistemas distribuidos, toda la comunicación se basa en el envío y la recepción (de bajo nivel) de mensajes. Cuando el proceso A debe comunicarse con el proceso B, primero elabora un mensaje en su propio espacio de dirección. Después ejecuta una llamada de sistema y ocasiona que el sistema operativo envíe el mensaje sobre la red hacia B. Aunque esta idea básica parece bastante sencilla, para evitar el caos, A y B deben acordar el significado de los bits por enviar. Si A envía una nueva y brillante novela escrita en francés, y codificada con el código de caracteres EBCDIC de IBM, y B espera el inventario de un supermercado escrito en inglés, y codificado en ASCII, la comunicación será menos que óptima. El modelo OSI está diseñado para permitir que los sistemas abiertos se comuniquen. Un sistema abierto, es aquel que está preparado para comunicarse con cualquier otro sistema abierto mediante reglas estándar que regulen formato, contenido, y significado de los mensajes enviados y recibidos. Estas reglas están formalizadas en lo que conocemos como protocolos. Para permitir a un grupo de computadoras comunicarse a través de una red, deben acordarse los protocolos a utilizar. Se hace una diferencia entre dos tipos generales de protocolos. Con los protocolos orientados a conexión, antes de intercambiar datos, el remitente y el destinatario primero establecen explícitamente una conexión, y posiblemente negocian el protocolo que utilizarán. El teléfono es un sistema de comunicación orientado a conexión. Con los protocolos orientados a no conexión, no se necesita una configuración por adelantado. El remitente sólo transmite el primer mensaje cuando está listo. Dejar una carta en el buzón es un ejemplo de comunicación orientada a no conexión. Con las computadoras, tanto la comunicación orientada a conexión como la que no son orientadas a conexión son comunes.

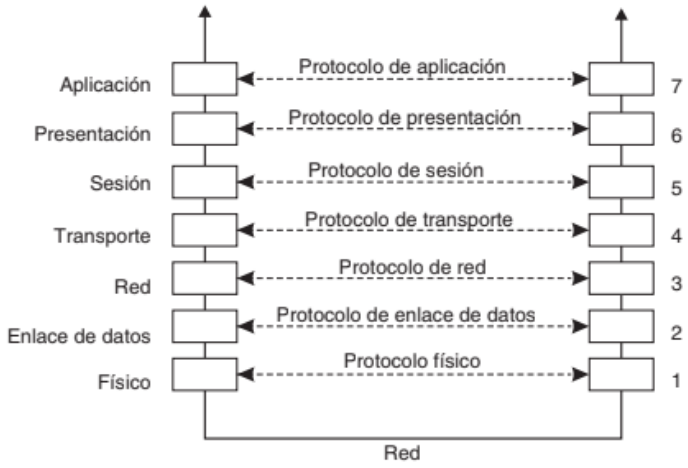

Figura 4-1. Capas, interfaces y protocolos del modelo OSI.

Protocolos de bajo nivel

Iniciemos con la explicación de las tres capas más bajas de la suite de protocolos de OSI. Juntas, estas capas implementan las funciones básicas que forman una red de computadoras. La capa física se ocupa de transmitir ceros y unos. Cuántos volts se utilizan para un 0 y cuántos para un 1, cuántos bits por segundo se pueden enviar, y si la transmisión puede ocurrir simultáneamente en ambas direcciones, son cuestiones clave de la capa física. Además, el tamaño y la forma del conector de red (plug), así como el número de pins y el significado de cada pin, son importantes aquí. El protocolo de la capa física maneja la estandarización eléctrica y mecánica y la señalización de interfaces, de tal modo que cuando una máquina envía un bit 0, éste en realidad se recibe como un bit 0, y no como un bit 1. Se han desarrollado muchos estándares para la capa física (para diferentes medios), por ejemplo, el estándar RS-232-C para líneas de comunicación en serie. La capa física sólo envía bits. Mientras no ocurran errores, todo está bien. Sin embargo, las redes de comunicación reales son propensas a errores, por lo que se necesita algún mecanismo para detectarlos y corregirlos. Este mecanismo es la tarea principal de la capa de enlace de datos; lo que hace es agrupar los bits en unidades, algunas veces llamadas tramas, y ve que cada trama se reciba correctamente. En una LAN, por lo general no hay necesidad de que el remitente localice al destinatario. Éste simplemente coloca el mensaje en la red y el destinatario lo recibe. Sin embargo, una red de área amplia consiste en una gran cantidad de máquinas, cada una con cierto número de líneas hacia otras máquinas, parecida a un mapa a gran escala con las principales ciudades y caminos que las conectan. Para que un mensaje llegue del remitente al destinatario, es probable que deba realizar una serie de saltos, y elegir en cada salto una línea de salida. La cuestión de cómo elegir la mejor ruta se conoce como enrutamiento, y es básicamente la tarea principal de la capa de red. El problema es complicado, debido a que la ruta más corta no siempre resulta ser la mejor. Lo que realmente importa es el monto del retraso en una ruta dada, lo cual, a su vez, se relaciona con el tráfico y el número de mensajes ubicados en la cola para transmitir por las diversas líneas. El retraso puede cambiar entonces con el transcurso del tiempo. Algunos algoritmos de enrutamiento intentan adaptarse a cargas cambiantes, mientras que otros se conforman con tomar decisiones basadas en promedios de largo plazo.

Protocolos de transporte

La capa de transporte forma la última parte de lo que podríamos llamar una pila básica de protocolos de red en el sentido de que implementa todos aquellos servicios no proporcionados en la interfaz de la capa de red, pero que son razonablemente necesarios para construir aplicaciones de red. En otras palabras, la capa de transporte transforma la red subyacente en algo que un desarrollador de aplicaciones puede utilizar. Los paquetes pueden perderse en el camino del remitente hacia el destinatario. Aunque algunas aplicaciones pueden manejar su propia recuperación de errores, otras prefieren tener una conexión confiable. El trabajo de la capa de transporte es proporcionar este servicio. La idea es que la capa de aplicación debe poder entregar un mensaje a la capa de transporte con la expectativa de que será entregado sin pérdidas. Una vez que recibe un mensaje desde la capa de aplicación, la capa de transporte lo divide en piezas lo suficientemente pequeñas como para transmitirlas, asigna a cada pieza un número secuencial, y después las envía todas. El análisis efectuado en el encabezado de la capa de transporte tiene que ver con qué paquetes se han enviado, cuáles se han recibido, para cuántos más tiene espacio el destinatario, cuáles deben retransmitirse, y otros aspectos similares.

Protocolos de más alto nivel

Por encima de la capa de transporte, OSI diferenció tres capas adicionales. En la práctica, únicamente la capa de aplicación se utiliza siempre. De hecho, en la suite de protocolos de internet, lo que se encuentra por encima de la capa de transporte se agrupa todo junto. De cara a los sistemas middleware, en esta sección veremos que ni el enfoque OSI ni el de internet son realmente adecuados. La capa de sesión es básicamente una versión mejorada de la capa de transporte. La capa de sesión proporciona control de diálogo para dar seguimiento a la parte que está comunicando en el momento, también proporciona herramientas de sincronización. Éstas son útiles para que los usuarios inserten puntos de verificación en transferencias largas, de tal modo que cuando sucede un desperfecto, sólo es necesario regresar al último punto de verificación, y no hasta el comienzo. En la práctica, pocas aplicaciones se interesan en la capa de sesión, y raramente es soportada. Incluso no está presente en la suite de protocolos de internet. Sin embargo, en el contexto de desarrollar soluciones middleware, el concepto de una sesión y sus protocolos relacionados se han vuelto muy importantes, principalmente cuando definen protocolos de comunicación de más alto nivel.

Protocolos middleware

El middleware es una aplicación que lógicamente reside (la mayor parte del tiempo) en la capa de aplicación, pero que contiene muchos protocolos de propósito general que garantizan sus propias capas, independientemente de otras aplicaciones más específicas. Podemos establecer cierta diferencia entre los protocolos de comunicación de alto nivel y los protocolos implementados para establecer diversos servicios middleware. Existen numerosos protocolos para soportar toda una variedad de servicios middleware. Por ejemplo, como explicaremos en el capítulo 9, hay varias formas de establecer la autenticación, es decir, proporcionar evidencia de una identidad declarada. Los protocolos de autenticación no están muy relacionados con alguna aplicación específica, pero pueden estar integrados en un sistema middleware como un servicio general. De igual manera, los protocolos de autorización mediante los cuales se garantiza el acceso a usuarios y procesos autenticados, a los recursos para los que tienen autorización, tienden a ser de naturaleza general e independientes de aplicaciones. Los protocolos de comunicación middleware soportan servicios de comunicación de alto nivel. Por ejemplo, en las dos siguientes secciones explicaremos protocolos que permiten que un proceso llame a un procedimiento o invoque a un objeto ubicado en una máquina remota de manera muy transparente. De igual forma, existen servicios de comunicación de alto nivel para establecer y sincronizar flujos para transferencia de datos en tiempo real, tales como los necesarios para aplicaciones multimedia. Como un último ejemplo, algunos sistemas middleware ofrecen servicios confiables de multidifusión que escalan a miles de destinatarios esparcidos en una red de área amplia. Algunos de los protocolos de comunicación middleware podrían pertenecer a la capa de transporte, pero es posible que existan razones específicas para mantenerlos a un nivel más alto. Por ejemplo, los confiables servicios de multidifusión que garantizan la escalabilidad pueden implementarse sólo cuando se consideran los requerimientos de la aplicación. En consecuencia, un sistema middleware puede ofrecer diferentes protocolos (ajustables), que a su vez se implementan utilizando diferentes protocolos de transporte, pero ofreciendo una sola interfaz.

Tipos de comunicación

Para comprender las diversas alternativas en comunicación que ofrece el middleware a las aplicaciones, veremos al middleware como un servicio adicional en el cómputo cliente-servidor, tal como ilustra la figura 4-4. Por ejemplo, considere un sistema de correo electrónico. En principio, la parte central del sistema de entrega de correo puede considerarse como un servicio de comunicación middleware. Cada servidor ejecuta un agente usuario que permite a los usuarios redactar, enviar, y recibir correo electrónico. Un agente usuario de envío pasa tal correo al sistema de entrega de correo, esperando a su vez entregar el correo al destinatario en algún momento. De igual manera, el agente usuario presente del lado del destinatario se conecta al sistema de entrega de correo para ver si ha llegado algún correo. Si es así, los mensajes se transfieren al agente usuario de tal modo que puedan desplegarse y ser leídos por el usuario.

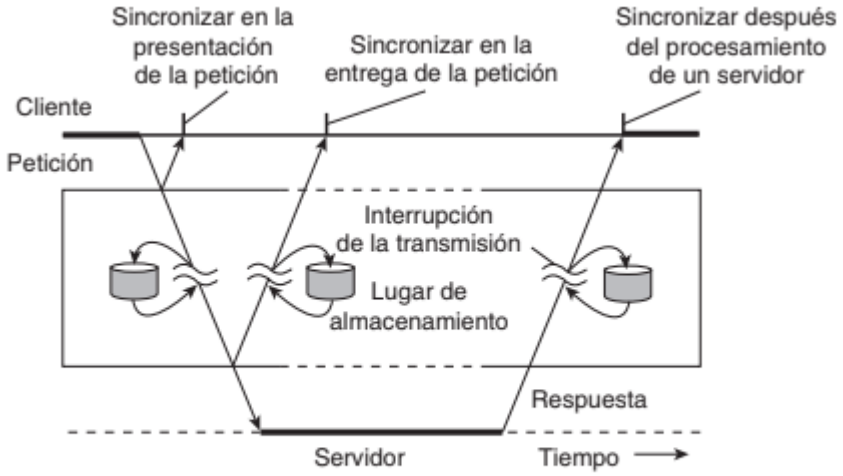

Figura 4-4. Perspectiva del middleware como un servicio intermediario (distribuido) al nivel de comunicación de aplicaciones.

Llamas a procedimientos remotos

Muchos sistemas distribuidos se han basado en el intercambio explícito de mensajes entre procesos. Sin embargo, los procedimientos send y receive no ocultan en absoluto la comunicación, lo cual es importante para lograr la transparencia en los sistemas distribuidos. Este problema, se conoce desde hace mucho, pero poco se hizo hasta que un artículo escrito por Birrell y Nelson (1984) mostró una forma completamente diferente de manejar la comunicación. Aunque la idea es mantenerla simple (una vez que alguien ha pensado en ello), con frecuencia las implicaciones son sutiles. En esta sección examinaremos el concepto, su implementación, sus fortalezas, y sus debilidades. En pocas palabras, lo que Birrell y Nelson sugirieron fue permitir que los programas llamaran a procedimientos ubicados en otras máquinas. Cuando un proceso de la máquina A llama a un procedimiento de la máquina B, el proceso que llama desde A se suspende, y la ejecución del procedimiento llamado ocurre en B. La información puede transportarse en los parámetros desde quien llama hasta el que es llamado, y puede regresar en el procedimiento resultante. Ningún mensaje de paso es visible para el programador. Este método se conoce como llamada a procedimiento remoto, o simplemente RPC.

Operación básica RPC

Primero iniciamos con una explicación de las llamadas a procedimientos convencionales, y después explicamos cómo la propia llamada se divide en una parte cliente y una parte servidor y que cada parte se ejecuta en máquinas diferentes.

Llamada a un procedimiento convencional

Para comprender cómo funciona la RPC, es importante que primero comprendamos totalmente cómo funciona una llamada a un procedimiento convencional (es decir, en una sola máquina). Considere una llamada en C como count read(fd, buf, nbytes); donde fd es un entero que indica un archivo, buf es un arreglo de caracteres en el que se leen los datos, y nbytes es otro entero que indica cuántos bytes leer. Si la llamada se hace desde el programa principal, la pila será como la que muestra la figura 4-5(a) antes de la llamada. Para realizar la llamada, el procedimiento que llama coloca los parámetros en la pila, en un orden del último al primero, como ilustra la figura 4-5(b). (La razón por la que los compiladores de C introduzcan los parámetros en orden inverso tiene que ver con printf —ya que, al hacerlo, printf siempre puede localizar a su primer parámetro, la cadena de formato.) Después que el procedimiento read ha terminado su ejecución, coloca el valor de retorno en un registro, elimina la dirección de retorno, y devuelve el control al procedimiento que llama. Este último elimina entonces los parámetros de la pila y la devuelve al estado original que tenía antes de la llamada.

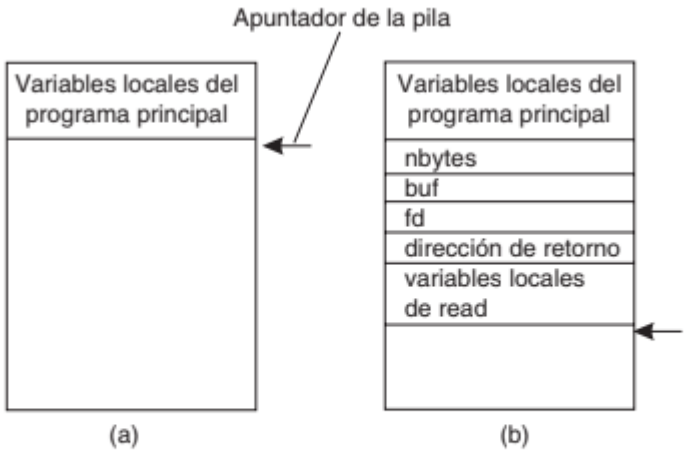

Figura 4-5. (a) Paso de parámetros en una llamada a un procedimiento local: la pila antes de la llamada a read. (b) La pila mientras el procedimiento llamado está activo.

Resguardos del cliente y servidor

La idea tras la RPC es hacerla parecer como una llamada local. En otras palabras, deseamos que la RPC sea transparente —el procedimiento de llamada no debe advertir que el procedimiento llamado se ejecuta en una máquina diferente o viceversa—. Suponga que un programa necesita leer algunos datos desde un archivo. El programador coloca en el código una llamada a read para obtener los datos. En un sistema tradicional (de un solo procesador), la rutina read se extrae de la biblioteca mediante el enlazador y se inserta en el programa objeto. Éste es un procedimiento corto, y generalmente se implementa llamando a un sistema read equivalente. En otras palabras, el procedimiento read es un tipo de interfaz establecido entre el código de usuario y el sistema operativo local. La RPC logra su transparencia de manera análoga. Cuando read es realmente un procedimiento remoto (por ejemplo, uno que se ejecutará en un archivo de la máquina servidor), se coloca en la biblioteca una versión diferente de read, llamada resguardo del cliente. Como al original, a éste se le llama utilizando también la secuencia de la figura 4-5(b). Además, como el original, éste hace igualmente una llamada al sistema operativo local. Sólo a diferencia del original, éste no solicita datos al sistema operativo. En vez de eso, empaca los parámetros en un mensaje y solicita que el mensaje sea enviado al servidor, según muestra la figura 4-6. Si seguimos la llamada a send, la matriz cliente llama a receive y se bloquea a sí misma hasta que la respuesta regresa.

Figura 4-6. Principio de la RPC entre un programa cliente y servidor.

Paso de parámetros

La función del resguardo del cliente es tomar sus parámetros, empacarlos en un mensaje, y enviarlos a la matriz servidor. Esto parece sencillo, pero no es tanto como puede parecer. En esta sección veremos algunas de las cuestiones relacionadas con el paso de parámetros en sistemas RPC.

Paso de parámetros de valor

Empacar parámetros en un mensaje se conoce como ordenamiento de parámetros. Como un ejemplo muy sencillo, considere un procedimiento remoto, add(i, j), que toma dos parámetros enteros, i y j, y devuelve la suma aritmética como resultado. (Como una cuestión práctica, uno no realizaría un procedimiento remoto tan simple debido a la sobrecarga, pero como ejemplo, lo haremos.) La llamada a add, aparece en el lado izquierdo (en el proceso cliente) de la figura 4-7. El resguardo del cliente toma sus dos parámetros y los coloca en un mensaje como se indica. También coloca el nombre o el número del procedimiento por llamar en el mensaje porque el servidor puede soportar muchas llamadas diferentes, y se le debe informar cuál llamada es requerida.

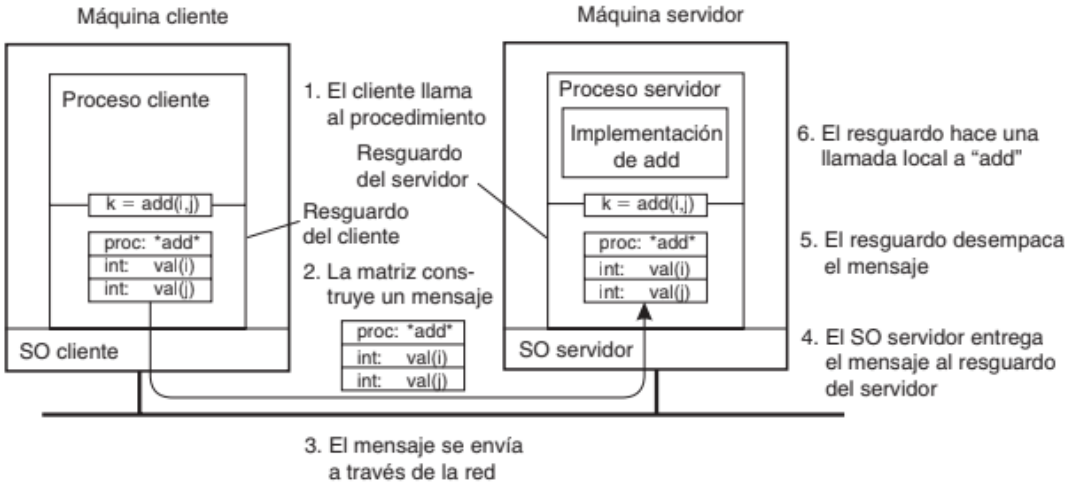

Figura 4-7. Pasos involucrados en un cálculo remoto a través de RPC.

RPC asíncrona

Igual que en las llamadas a procedimientos convencionales, cuando un cliente llama a un procedimiento remoto, el cliente se bloqueará hasta que se devuelva una respuesta. El comportamiento estricto de solicitud-respuesta es innecesario cuando no hay un resultado por devolver, y sólo conduce a bloquear al cliente mientras pudo haber continuado y realizar trabajo útil antes de la petición de llamada a un procedimiento remoto. Ejemplos de que a menudo no hay necesidad de esperar una respuesta incluyen: la transferencia de dinero de una cuenta a otra, agregar entradas a una base de datos, iniciar servicios remotos, el procesamiento por lotes, etcétera. Para soportar tales situaciones, los sistemas RPC pueden proporcionar facilidades para lo que conocemos como RPC asíncronas, mediante las cuales, un cliente continúa trabajando de inmediato después de emitir la petición RPC. Con RPC asíncronas, al momento en que recibe la petición de RPC, el servidor envía inmediatamente una respuesta hacia el cliente y después llama al procedimiento solicitado. La respuesta representa un acuse de recibo para el cliente de que el servidor va a procesar la RPC. El cliente continuará su trabajo sin mayor bloqueo tan pronto reciba el acuse del servidor. La figura 4-10(b) muestra cómo interactúan el cliente y el servidor en el caso de RPC asíncronas. Para comparar, la figura 4-10(a) muestra el comportamiento normal de una solicitud-respuesta.

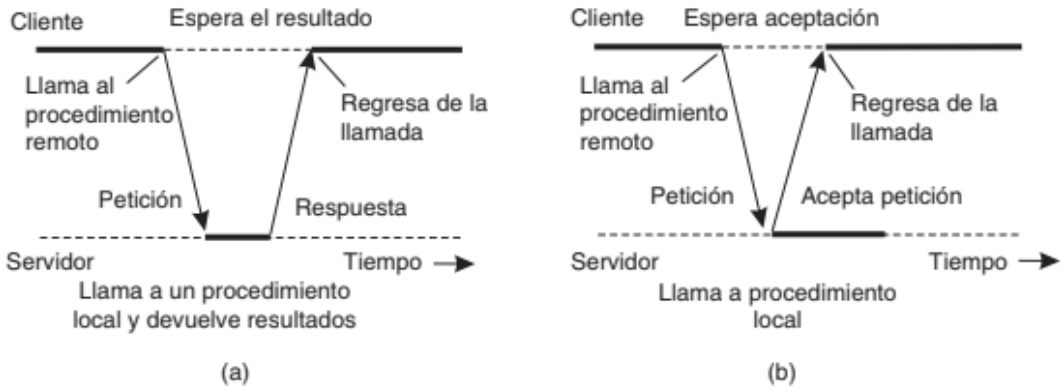

Figura 4-10. (a) Interacción entre cliente y servidor en una RPC tradicional. (b) Interacción utilizando RPC asíncronas.

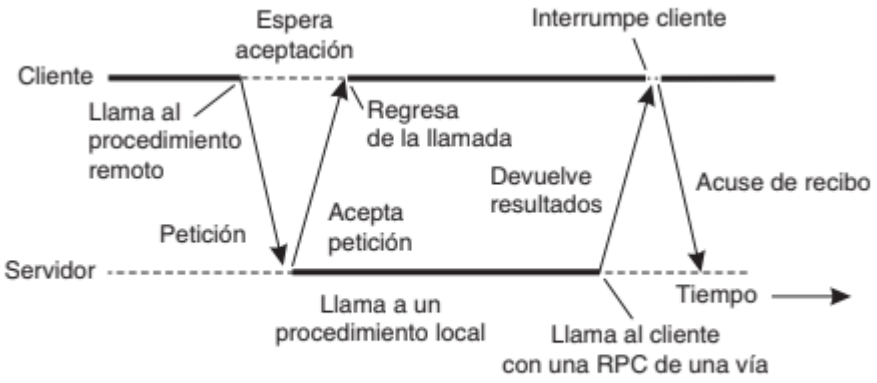

Figura 4-11. Cliente y servidor interactúan mediante dos RPC asíncronas.

Es importante observar que existen variantes de RPC asíncronas en las que el cliente continúa con su ejecución inmediatamente después de enviar la petición al servidor. En otras palabras, el cliente no espera un acuse del servidor por haber aceptado la petición. A una RPC como ésta la conocemos como RPC de una vía. El problema con este método es que cuando no se garantiza la confiabilidad, el cliente no puede saber con certeza si su petición será o no procesada. A estas cuestiones volveremos en el capítulo 8. De igual manera, en el caso de una RPC síncrona diferida, el cliente puede sondear al servidor para ver si los resultados están disponibles en lugar de dejar que el servidor le devuelva la llamada.

Ejemplo: DCE RPC

Las llamadas a procedimientos remotos se han adoptado ampliamente como la base del middleware y de los sistemas distribuidos en general. En esta sección, analizaremos con detalle un sistema específico RPC: el Ambiente de Computación Distribuida (DCE, por sus siglas en inglés), el cual fue desarrollado por Open Software Foundation (OSF), ahora conocida como The Open Group. DCE RPC no es tan popular como otros sistemas RPC, por ejemplo, Sun RPC. Sin embargo, DCE RPC es representativo de otros sistemas RPC, y sus especificaciones se adoptaron en el sistema base de Microsoft para computación distribuida, DCOM (Eddon y Eddon, 1998). Iniciamos con una breve introducción a DCE, después de lo cual consideraremos las funciones principales de DCE RPC.

Introducción a DCE

DCE es un verdadero sistema middleware porque está diseñado para ejecutarse como una capa de abstracción entre sistemas operativos existentes (redes) y aplicaciones distribuidas. Inicialmente fue diseñado para UNIX, pero ahora se ha llevado a los sistemas operativos principales, incluso VMS y variantes de Windows, así como a sistemas operativos de escritorio. La idea es que el cliente pueda tomar una colección de máquinas existentes, agregar el software DCE, y después ejecutar aplicaciones distribuidas, todo sin molestar a las aplicaciones existentes (no distribuidas). Aunque la mayoría de los paquetes DCE se ejecutan en espacio de usuario, en algunas configuraciones es necesario agregar una pieza (parte del sistema distribuido de archivos) al kernel. El propio Open Group sólo vende código fuente, el cual los distribuidores integran a sus sistemas. El modelo de programación subyacente a todo el DCE es el modelo cliente-servidor, el cual analizamos ampliamente en el capítulo anterior. Los procesos de usuario actúan como clientes para acceder a servicios remotos provistos mediante procesos servidor. Algunos de estos servicios son parte del propio DCE, pero otros, pertenecen a las aplicaciones y son escritos por los programadores de las aplicaciones. Toda comunicación entre clientes y servidores ocurre mediante RPC.

Objetivos del DCE RPC

Los objetivos del sistema DCE RPC son relativamente tradicionales. El primero y más importante objetivo es que el sistema RPC haga posible que un cliente acceda a un servicio remoto mediante una simple llamada a un procedimiento local. Esta interfaz posibilita la escritura de programas cliente (es decir, aplicaciones) de manera sencilla y conocida para la mayoría de los programadores. Depende del sistema RPC ocultar todos los detalles a los clientes, y hasta cierto punto, también a los servidores. Para comenzar, el sistema RPC puede localizar automáticamente el servidor correcto, y posteriormente establecer la comunicación entre el software cliente y servidor (en general, conocida como vinculación). También puede manejar el transporte de mensajes en ambas direcciones, fragmentarlos y reensamblarlos, según sea necesario (por ejemplo, si uno de los parámetros es un arreglo grande). Por último, el sistema RPC puede manejar automáticamente conversiones de tipos de datos entre el cliente y el servidor, incluso cuando se ejecutan en diferentes arquitecturas y tienen distintos ordenamientos de bytes.

Cómo escribir un cliente y un servidor

El sistema DCE RPC consiste en una cantidad de componentes que incluye lenguajes, bibliotecas, demonios, y programas de utilerías, entre otros; y juntos hacen posible escribir clientes y servidores. En esta sección describiremos las piezas y cómo juntarlas. La figura 4-12 muestra un resumen de todo el proceso de escribir y utilizar un RPC cliente y servidor. En un sistema cliente-servidor, el pegamento que mantiene todo unido es la definición de interfaz, tal como se especifica en el Lenguaje de Definición de Interfaces, o IDL por sus siglas en inglés. El IDL permite implementar declaraciones de procedimientos en una forma muy parecida a los prototipos de funciones en ANSI C. Los archivos IDL también pueden contener definiciones de tipos, declaraciones de constantes, y otra información necesaria para organizar parámetros de manera correcta y desorganizar resultados. Idealmente, la definición de interfaz también debe contener una definición formal de lo que hacen los procedimientos, pero tal definición cae más allá del actual estado de cosas, por ello sólo define la sintaxis de las llamadas, no su semántica. Cuando mucho, quien escribe puede agregar algunos comentarios que describen lo que hacen los procedimientos.

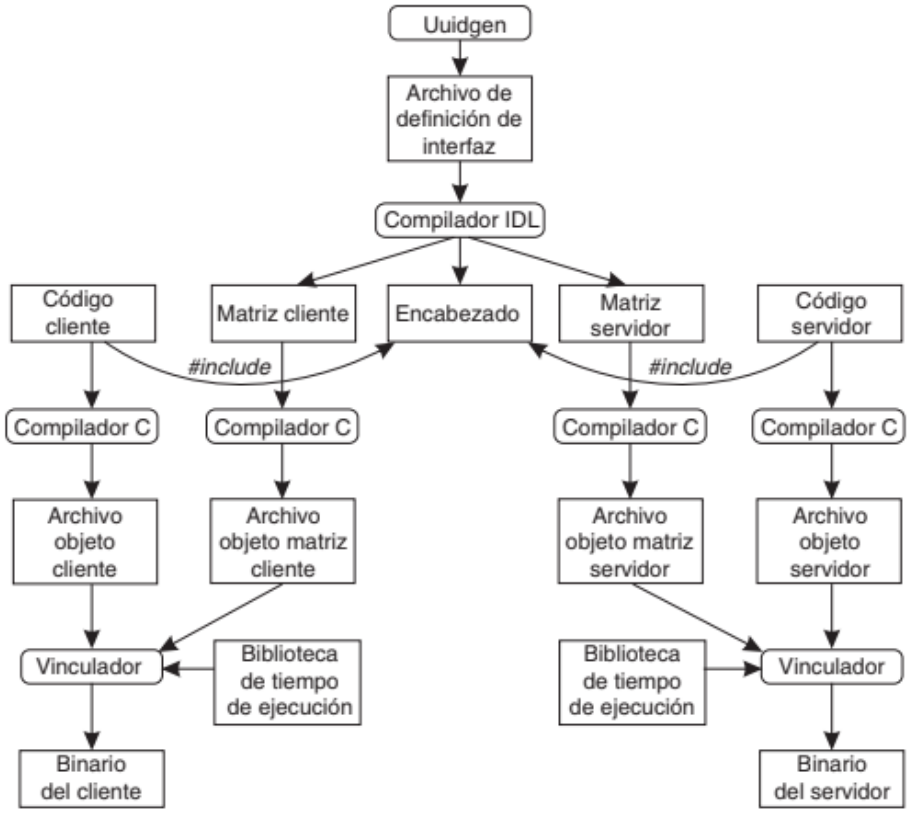

Figura 4-12. Pasos para escribir un cliente y un servidor en DCE RPC.

Vinculación de un cliente con un servidor

Con el propósito de permitir que un cliente llame a un servidor, es necesario que el servidor se registre y prepare para aceptar llamadas entrantes. El registro de un servidor hace posible que un cliente localice al servidor y se vincule con él. La localización de un servidor se hace en dos pasos:

1. Ubicación de la máquina servidor.

2. Ubicación del servidor (es decir, el proceso correcto) en esa máquina.

El segundo paso es de algún modo sutil. Básicamente, se trata de que, para comunicarse con un servidor, el cliente necesita conocer un punto final —ubicado en la máquina del servidor— al cual pueda enviar mensajes. El sistema operativo del servidor utiliza un punto final (también conocido generalmente como puerto) para diferenciar mensajes de entrada para diferentes procesos. En DCE, se mantiene una tabla de pares (servidor, punto final) en cada máquina servidor mediante un proceso llamado demonio DCE. Antes de estar disponible para peticiones de entrada, el servidor debe solicitar al sistema operativo un punto final; después registra este punto final con el demonio DCE que, a su vez, registra esta información (incluyendo los protocolos a los que se refiere el servidor) en la tabla de puntos finales para uso futuro.

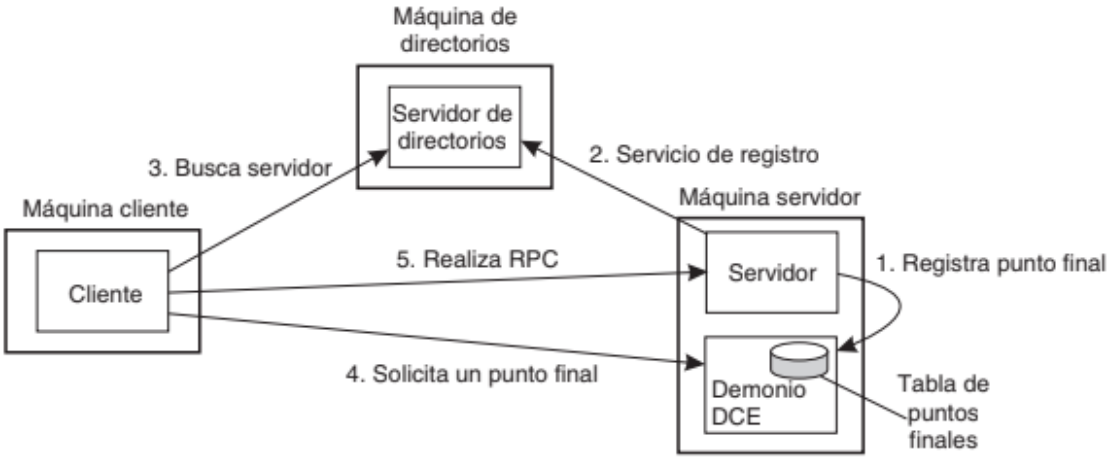

Figura 4-13. Vinculación cliente-servidor en DCE.

Comunicación orientada a mensajes

En los sistemas distribuidos, las llamadas a procedimientos remotos y las invocaciones a objetos remotos contribuyen a ocultar la comunicación, es decir, mejoran la transparencia de acceso. Por desgracia, ningún mecanismo es siempre adecuado. En particular, cuando no es posible asumir que el lado receptor está en ejecución al momento en que se hace una petición, se necesitan servicios de comunicación alternos. De igual manera, la naturaleza síncrona de las RPC, por la cual un cliente se bloquea hasta que su petición es procesada, algunas veces necesita remplazarse con algo más.

Comunicación transitoria orientada a mensajes

Muchos sistemas y aplicaciones distribuidos se construyen directamente sobre el sencillo modelo orientado a mensajes ofrecido por la capa de transporte. Para comprender esto y apreciar a los sistemas orientados a mensajes como parte de las soluciones middleware, primero explicaremos la mensajería a través del nivel de transporte mediante sockets.

Sockets Berkeley

Se ha puesto especial atención a la estandarización de la interfaz de la capa de transporte para permitir a los programadores utilizar la suite completa de los protocolos (de mensajería) mediante un simple conjunto de primitivas. Además, las interfaces estándar facilitan llevar una aplicación a una máquina diferente. La primitiva bind asocia una dirección local con el socket creado recientemente. Por ejemplo, un servidor debe enlazar la dirección IP de su máquina con un número de puerto (probablemente muy conocido) para un socket. El enlace le indica al sistema operativo que el servidor sólo desea recibir mensajes en la dirección y el puerto especificados.

Primitiva	Significado
Socket	Crea un nuevo punto final de comunicación
Bind	Asocia una dirección local a un socket
Listen	Anuncia la conveniencia de aceptar conexiones
Accept	Bloquea a quien llama hasta que llega una petición de conexión
Connect	Activa el intento de establecer una conexión
Send	Envía algunos datos a través de la conexión
Receive	Recibe algunos datos a través de la conexión
Close	Libera la conexión

Figura 4-14. Primitivas de socket para TCP/IP.

La interfaz de paso de mensajes (MPI)

Con la llegada de multicomputadoras de alto rendimiento, los desarrolladores han buscado primitivas orientadas a mensajes que les permitan escribir fácilmente aplicaciones altamente eficientes. Esto significa que las primitivas deben estar en un nivel de abstracción conveniente (para un desarrollo sencillo de aplicaciones), y que su implementación sólo implique una sobrecarga mínima.

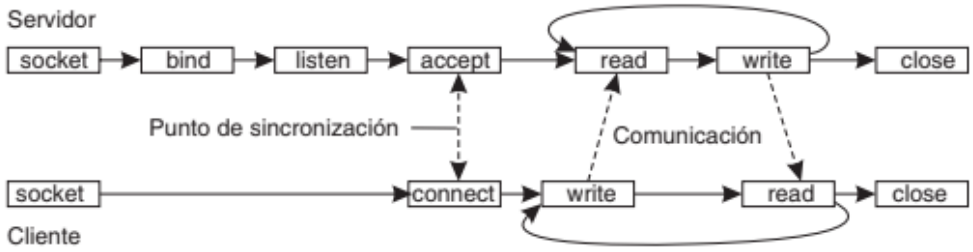

Figura 4-15. Patrón de comunicación orientada a conexiones mediante sockets.

La comunicación asíncrona transitoria se soporta mediante la primitiva MPI_bsend. El remitente presenta un mensaje para su transmisión, el cual, generalmente se copia primero en un bufer local de la MPI del sistema en ejecución. Cuando se ha copiado el mensaje, el remitente continúa. La MPI del sistema en ejecución eliminará el mensaje de su búfer local y se encargará de la transmisión tan pronto como un destinatario haya llamado a una primitiva de recepción.

Primitiva	Significado
MPI_bsend	Adjunta un mensaje de salida a un búfer local de envío
MPI_send	Envía un mensaje y espera hasta que es copiado en un búfer local o remoto
MPI_ssend	Envía un mensaje hasta que comienza la recepción
MPI_sendrecv	Envía un mensaje y espera una respuesta
MPI_isend	Pasa una referencia a un mensaje de salida, y continúa
MPI_issend	Pasa una referencia a un mensaje de salida, y espera hasta que inicia la recepción
MPI_recv	Recibe un mensaje; bloquea si no hay alguno
MPI_irecv	Verifica si hay algún mensaje de entrada, pero no bloquea

Figura 4-16. Algunas de las primitivas para paso de mensajes más conocidas de la MPI.

Comunicación persistente orientada a mensajes

Ahora, explicaremos una clase importante de servicios middleware orientados a mensajes, generalmente conocidos como sistemas de colas de mensajes, o simplemente Middleware Orientado a Mensajes (MOM, por sus siglas en inglés). Los sistemas de colas de mensajes proporcionan un amplio soporte para comunicación asíncrona persistente. La esencia de estos sistemas es que ofrecen capacidad de almacenamiento de término medio para mensajes, sin la necesidad de que el remitente o el destinatario estén activos durante la transmisión del mensaje. Una diferencia importante con los sockets Berkeley y la MPI, es que los sistemas de colas de mensajes están dirigidos al soporte de transferencias de mensajes que toman minutos en lugar de segundos o milisegundos. Primero explicaremos un método general de los sistemas de colas de mensajes, y concluiremos esta sección comparándolos con sistemas más tradicionales tales como los sistemas de correo electrónico de internet.

Modelo de colas de mensajes

La idea básica detrás de un sistema de colas de mensajes es que las aplicaciones se comunican insertando mensajes en colas específicas. Estos mensajes son reenviados a una serie de servidores de comunicación y en algún momento se entregan en su destino, incluso si éste no estaba disponible cuando se envió el mensaje. En la práctica, la mayoría de los servidores de comunicación están directamente conectados uno con otro. En otras palabras, por lo general, un mensaje se transfiere directamente hacia un servidor destino. En principio, cada aplicación tiene su propia cola privada a la que otras aplicaciones pueden enviar mensajes. Una cola puede ser leída sólo por su aplicación asociada, pero también es posible que varias aplicaciones compartan una sola cola. Un aspecto importante de los sistemas de colas de mensajes es que, por lo general, al remitente sólo se le garantiza que su mensaje se insertará en algún momento en la cola del destinatario. No se dan garantías sobre cuándo, o de si el mensaje en realidad será leído, lo cual es completamente definido por el comportamiento del destinatario.

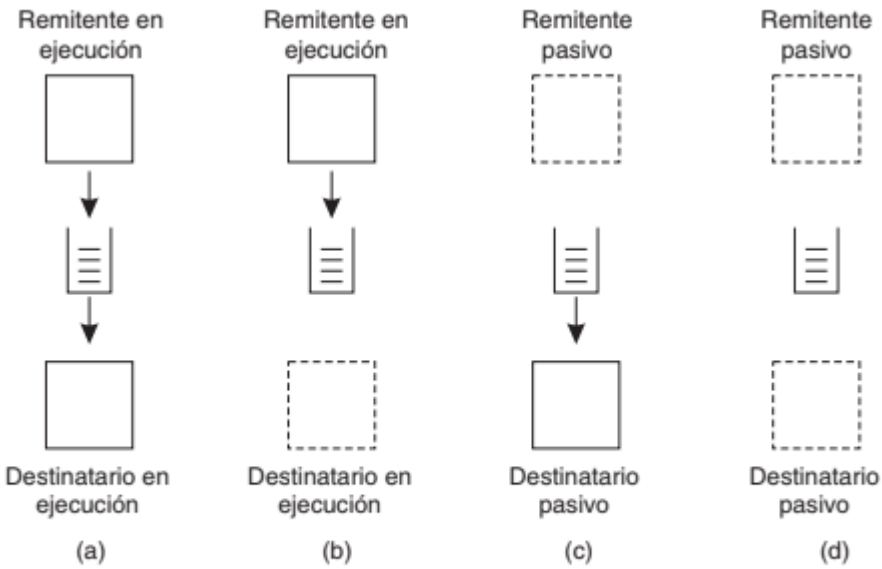

Figura 4-17. Cuatro combinaciones de comunicaciones muy poco acopladas mediante el uso de colas.

En la figura 4-17(a), tanto el remitente como el destinatario están en ejecución durante toda la transmisión de un mensaje. En la figura 4-17(b), sólo el remitente está en ejecución mientras el destinatario se encuentra pasivo, es decir, en un estado en el que no es posible la entrega del mensaje. Sin embargo, el remitente puede enviar mensajes. La combinación de un remitente pasivo y un destinatario en ejecución aparece en la figura 4-17(c). En este caso, el destinatario puede leer mensajes que le fueron enviados, pero no es necesario que sus respectivos remitentes se encuentren en ejecución. Por último, en la figura 4-17(d), vemos la situación en que el sistema almacena (y posiblemente transmite) mensajes, incluso mientras el remitente y los destinatarios se encuentran pasivos. En principio, los mensajes pueden contener cualquier información. El único aspecto importante por cubrir desde la perspectiva del middleware es que los mensajes se direccionen adecuadamente. En la práctica, el direccionamiento se hace proporcionando un nombre único de la cola de destino. En algunos casos el tamaño del mensaje puede limitarse, aunque también es posible que el sistema subyacente se encargue de fragmentar y ensamblar grandes mensajes de manera completamente transparente para las aplicaciones. Un efecto de este método, es que la interfaz básica ofrecida a las aplicaciones puede ser extremadamente simple, como indica la figura 4-18.

COMUNICACIÓN ORIENTADA A FLUJOS

La comunicación, como la hemos explicado hasta el momento, se ha concentrado en el intercambio más o menos independiente y completo de unidades de información. Algunos ejemplos incluyen una petición para invocar un procedimiento, la respuesta a tal petición, y mensajes intercambiados entre aplicaciones como en los sistemas de colas de mensajes. La principal característica de este tipo de comunicación es que no importa en qué punto del tiempo en particular ocurre la comunicación. Aunque un sistema puede comportarse de forma muy lenta o muy rápidamente, la sincronización no tiene efecto alguno sobre la integridad. También hay formas de comunicación en las que la sincronización juega un papel crucial. Por ejemplo, considere un flujo de audio concebido como una secuencia de muestras de 16 bits, donde cada muestra representa la amplitud de la onda sonora como se hace con la Modulación de Código de Pulso (PCM, por sus siglas en inglés). También suponga que el flujo de audio representa calidad de disco compacto, lo cual significa que la onda sonora original se ha muestreado a una frecuencia de 44100 Hz. Para reproducir el sonido original, es esencial que las muestras del flujo de audio se reproduzcan en el orden en que aparecen en el flujo, pero también en intervalos de exactamente 1/44100 segundos. La reproducción a una velocidad diferente producirá una versión incorrecta del sonido original.

Soporte para medios continuos

El soporte para intercambiar información dependiente del tiempo con frecuencia se conoce como soporte para medios continuos. Un medio se refiere al recurso mediante el cual se transmite la información. Estos recursos incluyen a los medios de almacenamiento y transmisión, medios de presentación como un monitor, etc. Un tipo importante de medio es la forma en que se representa la información. En otras palabras, ¿cómo se codifica la información en un sistema de cómputo? Se utilizan diferentes representaciones para diferentes tipos de información. Por ejemplo, el texto generalmente se codifica en ASCII o Unicode. Las imágenes pueden representarse en diferentes formatos, como GIF o JPEG. En un sistema de cómputo, los flujos de audio pueden codificarse al tomar muestras de 16 bits empleando PCM. En medios continuos, las relaciones temporales entre diferentes elementos de datos resultan fundamentales para interpretar correctamente lo que significan en realidad los datos. Ya dimos como ejemplo, la reproducción de una onda sonora mediante la reproducción de un flujo de audio. Como otro ejemplo, consideremos el movimiento.

Flujos y calidad del servicio

Los requerimientos de sincronización (y otros no funcionales) se expresan generalmente como requerimientos de Calidad del Servicio (QoS, por sus siglas en inglés). Estos requerimientos describen lo que se necesita del sistema distribuido subyacente y de la red para garantizar que, por ejemplo, las relaciones temporales de un flujo puedan preservarse. La QoS para flujos continuos de datos tiene que ver principalmente con puntualidad, volumen, y confiabilidad. En esta sección veremos la QoS y su relación con la configuración de un flujo. Mucho se ha dicho sobre cómo especificar la QoS requerida (por ejemplo, consulte a Jin y Nahrstedt, 2004). Desde la perspectiva de una aplicación, en muchos casos esto se reduce a especificar algunas propiedades importantes (Halsall, 2001):

1. La velocidad de bits requerida a la que deben transportarse los datos.
2. El retraso máximo hasta que se haya configurado una sesión (es decir, cuando una aplicación puede comenzar el envío de datos).
3. El retraso máximo fin a fin (es decir, cuánto tiempo le llevará a una unidad de datos llegar hasta un destinatario).
4. La varianza del retraso máximo, o inestabilidad.
5. El retraso máximo de un ciclo.

Resulta importante advertir que es posible realizar muchas mejoras a estas especificaciones tal como, por ejemplo, explican Steinmetz y Nahrstadt (2004). Sin embargo, cuando se trata de la comunicación orientada a flujos que se basa en el protocolo de pila de internet, simplemente debemos vivir con el hecho de que la base de la comunicación está formada por un servicio extremadamente simple de datagramas IP. Cuando lo anterior es un hecho, como fácilmente puede ser el caso en internet, especificar la IP permite a la implementación de un protocolo deshacerse de paquetes cuando lo juzgue conveniente.

Sincronización de flujos

Un tema muy importante en sistemas multimedia es que flujos diferentes, posiblemente en forma de un flujo complejo, están mutuamente sincronizados. La sincronización de flujos tiene que ver con mantener las relaciones temporales entre flujos. Ocurren dos tipos de sincronización. La forma de sincronización más sencilla sucede entre un flujo discreto de datos y un flujo continuo de datos. Por ejemplo, consideremos una diapositiva mejorada con audio y mostrada en la web. La diapositiva se transfiere del servidor al cliente en forma de un flujo de datos discreto. Al mismo tiempo, el cliente debe reproducir una parte específica de un flujo de audio que coincida con la diapositiva en curso y también sea traído desde el servidor. En este caso, el flujo de audio se sincroniza con la presentación de diapositivas. Un tipo más demandante de sincronización es el que tiene lugar entre flujos continuos de datos. Un ejemplo diario es la reproducción de una película en la que el flujo de video necesita sincronizarse con el de audio, proceso conocido como sincronización de labios. Otro ejemplo de sincronización es la reproducción de un flujo de audio estereofónico que consiste en dos subflujos, uno para cada canal. Una reproducción adecuada requiere que los dos subflujos estén muy bien sincronizados: una diferencia de más de 20 ms puede distorsionar el efecto estereofónico. La sincronización ocurre al nivel de las unidades de datos que conforman el flujo. En otras palabras, podemos sincronizar dos flujos sólo entre unidades de datos. La elección de la unidad de datos depende en gran medida del nivel de abstracción con que se considere al flujo de datos. Para concretar, consideremos nuevamente un flujo de audio (de un solo canal) de la calidad de un disco compacto. Con la granulación más fina, tal flujo aparece como una secuencia de muestras de 16 bits. Con una frecuencia de muestreo de 44100 Hz, la sincronización con otros flujos de audio podría, en teoría, ocurrir aproximadamente cada 23 μs. Para efectos estereofónicos de más alta calidad, es evidente que se necesita la sincronización a este nivel.

COMUNICACIÓN POR MULTITRANSMISIÓN

Un tema importante en comunicación de sistemas distribuidos es el soporte para enviar datos a varios destinatarios, lo cual también se conoce como comunicación por multitransmisión. Por muchos años, este aspecto ha pertenecido al dominio de los protocolos de red, donde se han implementado y evaluado diversas propuestas de solución al nivel de red y al nivel de transporte (Janic, 2005; y Obraczka, 1998). Un asunto importante en todas las soluciones fue configurar rutas de comunicación para diseminar información. En la práctica, esto involucró un esfuerzo enorme de administración que en muchos casos requería la intervención humana. Además, como no hay propuestas que coincidan, los ISP se han mostrado reacios a soportar la multitransmisión (Diot y cols., 2000). Con la llegada de la tecnología de punto a punto, y la notablemente estructurada administración sobrepuesta, se volvió más sencillo configurar rutas de comunicación. Debido a que las soluciones de punto a punto se utilizan típicamente en la capa de aplicación, se han introducido varias técnicas de multitransmisión al nivel de aplicación. En esta sección analizaremos brevemente estas técnicas.

Multitransmisión al nivel de aplicación

La idea básica de la multitransmisión al nivel de aplicación es que los nodos se organizan en una red sobrepuesta, la cual después se utiliza para diseminar la información a sus miembros. Una observación importante es que los ruteadores de red no están organizados en grupos de miembros. En consecuencia, las conexiones entre los nodos de la red sobrepuesta pueden cruzar diversas uniones físicas, y como tal, los mensajes de enrutamiento dentro de la sobrepuesta pueden no ser óptimos en comparación con lo que se hubiese logrado mediante un enrutamiento al nivel de red. Los nodos como P que han solicitado explícitamente la unión con el árbol de multitransmisión también son, por definición, promotores. El resultado de este esquema es que construimos un árbol de multitransmisión a través de la red sobrepuesta con dos tipos de nodos: promotores puros que actúan como ayudantes, y nodos que también son promotores pero que han solicitado explícitamente la unión con el árbol. La multitransmisión ahora es sencilla: un nodo simplemente envía un mensaje por multitransmisión hacia la raíz del árbol ejecutando nuevamente la operación LOO-KUP(mid), después de lo cual ese mensaje puede enviarse a lo largo del árbol. Observemos que esta descripción de alto nivel de la multitransmisión en Scribe no le hace justicia a su diseño original. Invitamos al lector interesado a que consulte los detalles, los cuales puede encontrar en Castro y colaboradores (2002).

Diseminación de datos basada en el gossip

Una técnica que cobra cada vez más importancia para implementar la diseminación de información se basa en el comportamiento epidémico. Al observar cómo se propagan las enfermedades entre la gente, los investigadores han estudiado desde hace mucho si técnicas tan simples pudieran desarrollarse para propagar información en sistemas distribuidos de gran escala. El objetivo principal de estos protocolos epidémicos es propagar rápidamente información entre una gran colección de nodos utilizando sólo información local. En otras palabras, no hay un componente central mediante el cual se coordine la diseminación de información. Para explicar los principios generales de estos algoritmos, suponemos que, para un elemento específico de datos, todas las actualizaciones se inician en un solo nodo. De esta manera, simplemente evitamos conflictos escritura-escritura. La siguiente presentación se basa en el clásico artículo de Demers y colaboradores, (1987), sobre algoritmos epidémicos. Una visión general reciente sobre diseminación epidémica de información puede encontrarse en Eugster y colaboradores, (2004).

Conclusión

En conclusión, la comunicación en los sistemas operativos es fundamental para garantizar un funcionamiento eficiente y coordinado entre los distintos procesos y dispositivos. Los mecanismos de comunicación entre procesos, como los semáforos, las señales y la memoria compartida, permiten una gestión efectiva de los recursos del sistema y contribuyen a evitar conflictos y errores. Estos sistemas de comunicación no solo optimizan el rendimiento, sino que también aseguran la estabilidad y seguridad del entorno operativo. A medida que los sistemas evolucionan, es crucial seguir mejorando estos métodos para adaptarse a las demandas crecientes de aplicaciones más complejas y sistemas distribuidos.

INSTITUTO TECNOLOGICO SUPERIOR DE SAN ANDRES TUXTLA

CARRERA:

ING.INFORMATICA

DOCENTE:

MARIA DE LOS ANGELES PELAYO VAQUERO

MATERIA:

SISTEMAS OPERATIVOS II

ACTIVIDAD:

ENSAYO

ALUMNO:

OSVANY JESUS FONSECA ABRAJAN

FECHA:

22/10/2024

Introducción

La sincronización del reloj es un proceso fundamental en los sistemas digitales y de telecomunicaciones que garantiza que todos los dispositivos, servidores o nodos en una red compartan la misma referencia temporal. Esta alineación precisa del tiempo es crítica para asegurar la coherencia y eficiencia en procesos distribuidos, permitiendo una comunicación fluida, la correcta secuenciación de eventos y la integridad de datos. En entornos como redes informáticas, sistemas financieros, aplicaciones industriales y telecomunicaciones, incluso pequeñas discrepancias en los relojes pueden causar fallos graves. Por ejemplo, en sistemas financieros, un desfase de milisegundos entre servidores podría provocar inconsistencias en transacciones; en redes de telecomunicaciones, podría alterar el orden de los paquetes transmitidos. Por ello, es esencial implementar mecanismos de sincronización, como NTP (Network Time Protocol) o PTP (Precision Time Protocol), para asegurar que todos los componentes operen en perfecta armonía temporal. La sincronización del reloj no solo optimiza el rendimiento, sino que también contribuye a la seguridad del sistema, permitiendo auditorías precisas y evitando conflictos en procesos críticos. En un mundo cada vez más interconectado, la sincronización temporal no es solo una opción, sino una necesidad para garantizar el correcto funcionamiento de infraestructuras modernas.

SINCRONIZACIÓN DEL RELOJ

En un sistema centralizado, el tiempo no es ambiguo. Cuando un proceso quiere saber la hora, realiza una llamada de sistema y el núcleo se la dice. Si un proceso A pregunta la hora, y después un proceso B pregunta la hora, el valor que obtiene B será mayor que (o probablemente igual que) el valor que obtuvo A; con certeza no será menor. En un sistema distribuido, lograr un acuerdo con respecto al tiempo no es algo trivial. Por un momento, y a manera de ejemplo, pensemos sólo en las implicaciones de carecer de un tiempo global en el programa make de UNIX. En general, en UNIX, los programas grandes se dividen en varios archivos fuente, de modo tal que un cambio en un archivo fuente sólo necesita compilar un archivo, y no todos. Si un programa consta de 100 archivos, el hecho de no tener que recompilar todo, debido a que un archivo se modificó, incrementa enormemente la velocidad a la que los programadores pueden trabajar. La forma en que make funciona es sencilla. Cuando el programador ha terminado de modificar todos los archivos fuente, ejecuta make, el cual analiza las fechas en que todos los archivos fuente y objeto se modificaron por última vez. Si el archivo fuente input.c tiene la hora 2151, y el archivo objeto correspondiente input. o tiene la hora 2150, make sabe que input.c ha cambiado desde que input.o se creó, y por tanto input.c debe recompilarse. Por otra parte, si output.c tiene la hora 2144 y output.o tiene la hora 2145, no es necesario recompilar. Así, make revisa todos los archivos fuente para ver cuál necesita recompilarse, y llama al compilador para que lo haga. Ahora imaginemos lo que podría pasar en un sistema distribuido en el que no existiera un acuerdo global con respecto al tiempo. Supongamos que output. o tiene la hora 2144 como indicamos arriba, y que poco después output.c se modifica, pero se le asigna la hora 2143 debido a que el reloj de su máquina está ligeramente retrasado, como ilustra la figura 6-1. Make no llamará al compilador. El programa binario ejecutable resultante entonces contendrá una mezcla de archivos objeto proveniente de las fuentes anteriores y de las fuentes nuevas; el programa probablemente fallará y el programador enloquecerá intentando comprender lo que está mal en el código.

Relojes físicos

Casi todas las computadoras tienen un circuito para dar seguimiento al tiempo. A pesar del amplio uso de la palabra “reloj” para hacer referencia a estos dispositivos, en realidad no son relojes en el sentido usual. Tal vez cronómetro sea una mejor palabra con la cual designarlos. Un cronómetro de computadora, en general, es un cristal de cuarzo mecanizado con precisión. Cuando este dispositivo se mantiene sujeto a tensión, los cristales de cuarzo oscilan en una frecuencia bien definida que depende del tipo de cristal, de la forma de su corte, y de la cantidad de tensión. Hay dos registros asociados con cada cristal, un contador y un registro de mantenedor. Cada oscilación del cristal disminuye el contador en uno. Cuando el contador llega a cero, se genera una interrupción y el contador se reinicia a partir del registro de mantenedor. De esta manera, es posible programar un cronómetro para generar una interrupción 60 veces por segundo, o a cualquier otra frecuencia deseada. A cada interrupción se le conoce como marca de reloj. Cuando un sistema se inicia, generalmente solicita al usuario la fecha y la hora, las cuales se convierten en el número de marcas posteriores a una fecha de inicio conocida y se almacena en memoria. La mayoría de las computadoras tiene una pila especial que respalda la RAM CMOS de tal modo que no es necesario introducir la fecha ni la hora en inicios posteriores. En cada marca del reloj, el procedimiento de servicio de interrupción agrega uno a la hora almacenada en memoria. De esta manera, el reloj (software) se mantiene actualizado. Con una sola computadora y un solo reloj, no importa mucho si este reloj está desfasado por una pequeña cantidad. Debido a que todos los procesos de la máquina utilizan el mismo reloj, serán internamente consistentes. Por ejemplo, si el archivo input.c tiene la hora 2151 y el archivo input. o tiene la hora 2150, make recompilará el archivo fuente, incluso si el reloj está desfasado por 2 y las horas reales son 2153 y 2152, respectivamente. Lo que importa en realidad son los tiempos relativos.

La mayoría de las empresas de energía eléctrica sincronizan el tiempo de sus relojes de 60 o 50 Hz con el UTC, por lo que cuando el BIH anuncia un segundo vacío, estas empresas elevan su frecuencia a 61 o 51 Hz por 60 o 50 segundos para adelantar todos los relojes de su área de distribución. Debido a que un segundo es un intervalo digno de atención para una computadora, un sistema operativo que necesite mantener el tiempo preciso durante cierto periodo de años debe tener un software especial para contar los segundos vacíos conforme sean anunciados (a menos que utilice la línea de energía para el tiempo, lo cual resulta en general demasiado rudo). El número total de segundos vacíos introducidos al UTC hasta el momento es de aproximadamente 30. Para proporcionar el UTC a la gente que necesita el tiempo exacto, el National Institute of Standard Time (NIST) opera una estación de radio de onda corta con las siglas de llamado WWV desde Fort Collins, Colorado. WWV emite un pulso corto al inicio de cada segundo UTC. La precisión de WWV es de aproximadamente 1 ms, pero debido a las fluctuaciones atmosféricas aleatorias que pueden afectar la longitud de la ruta de la señal, en la práctica la precisión no es mejor que 10 ms. En Inglaterra, la estación MSF, que opera desde Rugby, Warwickshire, proporciona un servicio similar al de las estaciones de otros países. Varios satélites terrestres también ofrecen un servicio UTC. El Geostationary Environment Operational Satellite puede proporcionar una precisión UTC de 0.5 ms, y algunos otros satélites lo hacen incluso mejor. Utilizar ya sea radio de onda corta o servicios satelitales requiere un conocimiento preciso de la posición relativa del emisor y del receptor, para compensar el retraso en la propagación de la señal. Los receptores de radio para WWV, GEOS, y otras fuentes UTC ya están comercialmente disponibles.

Sistema de posicionamiento global

Como un paso hacia la solución de problemas de sincronizado de relojes, primero consideraremos un problema relacionado, a saber, cómo determinar nuestra posición geográfica en cualquier parte del planeta. Este problema de posicionamiento se resuelve por sí mismo a través de un sistema distribuido altamente específico y dedicado llamado GPS (por sus siglas en inglés), y que significa sistema de posicionamiento global. El GPS es un sistema distribuido basado en un satélite puesto en órbita en 1978. Aunque ha sido utilizado principalmente en aplicaciones militares, en años recientes ha encontrado su camino para muchas aplicaciones civiles, principalmente en la navegación. Sin embargo, existen muchos más campos de aplicación. Por ejemplo, los teléfonos GPS ahora permiten a quienes llaman rastrear la posición del otro, una característica que puede ser extremadamente útil cuando se está perdido o en problemas. Este principio puede aplicarse fácilmente para rastrear otras cosas, incluso mascotas, niños, automóviles, naves, etc. Zogg (2002) presenta una excelente cobertura sobre el GPS. El GPS utiliza 29 satélites que circulan cada uno en una órbita situada a una altura aproximada de 20000 km. Cada satélite tiene hasta cuatro relojes atómicos que son calibrados regularmente desde estaciones especiales ubicadas en la Tierra. Un satélite transmite su posición de manera continua, y registra el tiempo de cada mensaje con su tiempo local. Esta transmisión permite a cada receptor localizado en la Tierra calcular exactamente su propia posición, empleando en principio sólo tres satélites. Para explicar esto, primero supongamos que todos los relojes, incluyendo el del receptor, están sincronizados. Hasta el momento hemos asumido que las mediciones son perfectamente exactas. Por supuesto, no lo son. Por una parte, el GPS no considera segundos vacíos. En otras palabras, existe una desviación sistemática del UTC, la cual a partir del 1 de enero de 2006 es de 14 segundos. Tal error puede compensarse fácilmente en el software. Sin embargo, existen muchas otras fuentes de error, comenzando con el hecho de que los relojes atómicos satelitales no siempre se encuentran en perfecta sincronía, la posición de un satélite no se conoce con precisión, el reloj del receptor tiene una exactitud finita, la velocidad de propagación de la señal no es constante (la velocidad de las señales disminuye cuando, por ejemplo, entran a la ionosfera), etc. Más aún, todos sabemos que la Tierra no es una esfera perfecta, lo que nos lleva a necesitar más correcciones.

RELOJES LÓGICOS

Hasta el momento hemos supuesto que la sincronización de relojes está naturalmente relacionada con el tiempo real. Sin embargo, también hemos visto que puede ser suficiente que cada nodo coincida con un tiempo actual, sin que el tiempo sea necesariamente el mismo que el tiempo real. Podemos ir un paso más adelante. Por ejemplo, para ejecutar make, resulta adecuado que dos nodos acuerden que input. o sea, actualizado por una nueva versión de input.c. En este caso, dar seguimiento a los otros eventos de cada uno (como producir una nueva versión de input.c) es lo que importa. Para estos algoritmos, es una convención referirse a los relojes como relojes lógicos. En un artículo clásico, Lamport (1978) mostró que, aunque la sincronización de relojes es posible, no necesita ser absoluta. Si dos procesos no interactúan, no es necesario que sus relojes sean sincronizados ya que la falta de sincronización no se notaría y, por tanto, no ocasionaría problemas. Más aún, señaló que lo generalmente importante no es que todos los procesos coincidan exactamente en el tiempo, sino que coincidan en el orden en que ocurren los eventos. En el ejemplo de make, lo que cuenta es si input.c es más antiguo o reciente que input. o, y no sus tiempos absolutos de creación.

Relojes lógicos de Lamport

Ahora veamos el algoritmo de Lamport propuesto para asignar tiempos a eventos. Consideremos los tres procesos delineados en la figura 6-9(a). Los procesos se ejecutan en diferentes máquinas, cada una con su propio reloj, avanzando a su propia velocidad. Como podemos ver en la figura, cuando el reloj ha hecho marca 6 veces en el proceso P1, éste ha hecho marca 8 veces en el proceso P2, y 10 veces en el proceso P3. Cada reloj avanza a velocidad constante, pero las velocidades son diferentes debido a las diferencias en los cristales. La solución de Lamport se deriva directamente a partir de la relación ocurrencia-anterior. Debido a que m3 salió en el 60, debe llegar en el 61 o después. Por tanto, cada mensaje lleva el tiempo de envío de acuerdo con el reloj del remitente. Cuando un mensaje llega y el reloj del destinatario muestra un valor anterior al tiempo en que el mensaje fue enviado, el destinatario rápidamente adelanta su reloj para estar una unidad adelante del tiempo de envío. En la figura 6-9(b) observamos que m3 ahora llega en el 61. De manera similar, m4 llega en el 70. Con el propósito de prepararnos para abordar la explicación sobre relojes vectoriales, formulemos este procedimiento de manera más precisa. En este punto, es importante diferenciar tres capas de software distintas, como ya vimos en el capítulo 1: la red, la capa middleware, y una capa de aplicación, según muestra la figura 6-10. Lo siguiente es típico de la capa middleware.

Conclusión

La sincronización del reloj es esencial para garantizar la integridad, coherencia y eficiencia en sistemas digitales y distribuidos. Desde transacciones financieras y telecomunicaciones hasta redes industriales y dispositivos IoT, contar con una referencia temporal precisa permite que los procesos ocurran en el orden correcto y minimiza riesgos de errores. Protocolos como NTP y PTP han demostrado ser herramientas clave para mantener esta alineación temporal, adaptándose a las necesidades de precisión que varían entre diferentes aplicaciones. En un entorno globalmente interconectado, donde incluso diferencias de milisegundos pueden tener consecuencias significativas, la sincronización del reloj no solo optimiza el rendimiento, sino que también refuerza la seguridad y la capacidad de auditoría. De cara al futuro, garantizar un tiempo uniforme y preciso será indispensable para soportar la creciente complejidad de las tecnologías emergentes y los sistemas distribuidos en constante evolución.

Referencias bibliográficas

(S/f). Google.com. Recuperado el 22 de octubre de 2024, de https://drive.google.com/file/d/1NrmIiOY_zErWP92AJYbeMGn9v4_XYN38/view

EVALUACION UNIDAD 2. SISTEMAS OPERATIVOS II

INGENIERIA INFORMATICA ITSSAT

Se ha registrado el correo del encuestado (**221u0501@alumno.itssat.edu.mx**) al enviar este formulario.

VALOR 40 %

NOMBRE Y APELLIDOS *

Osvany Jesus Fonseca Abrajan

GRUPO: *

510-A

El sistema operativo tiene una tabla de procesos que contiene entradas para almacenar valores de los registros de la CPU, mapas de memoria, archivos abiertos, información contable, privilegios, etc. * 5 puntos

- (●) V
- ( ) F

Existen básicamente dos métodos para implementar un paquete de hilos. El primer método es la construcción de una biblioteca de hilos que se ejecutan por completo en el modo de usuario; el segundo es que el kernel esté al tanto de los hilos y los pueda calendarizar. * 5 puntos

- (●) V
- ( ) F

Una desventaja importante de los hilos a nivel de usuario es que, al invocar una llamada de bloqueo de sistema, ésta bloqueará todo el proceso al cual pertenece el hilo, y entonces bloqueará todos los hilos presentes en dicho proceso. * 5 puntos

- (●) V
- ( ) F

La combinación de (a nivel usuario) hilos y LWP trabaja de la siguiente manera. El paquete de hilos tiene una rutina simple para calendarizar el siguiente hilo. Al crear un LWP (lo cual se hace por medio de una llamada de sistema), el LWP cuenta con su propia pila, y tiene la instrucción de ejecutar la rutina de calendarización en busca de un hilo para ejecución. Si existen varios LWP, entonces cada uno invoca al planificador. La tabla de hilos, utilizada para seguir la pista del conjunto de hilos actuales, es compartida por los LWP... * 5 puntos

- (●) V
- ( ) F

Un servidor concurrente no manipula por sí mismo la petición, pero la pasa a un hilo separado o a otro proceso, después de lo cual de inmediato queda en espera de la siguiente petición entrante. Un servidor multihilos es ejemplo de un servidor concurrente. Una implementación alternativa para un servidor concurrente es la de crear un nuevo proceso para cada nueva petición entrante. * 5 puntos

- (●) V
- ( ) F

Para operar una entidad, es necesario acceder a ella, de modo que se requiere tener un punto de acceso. Un punto de acceso es otra clase, pero especial, de entidad en un sistema distribuido. Al nombre de un punto de acceso se le llama dirección. A la dirección del punto de acceso de una entidad se le denomina dirección de dicha entidad. * 5 puntos

- (●) V
- ( ) F

Un identificador verdadero es un nombre que tiene las propiedades siguientes: * 5 puntos

1. Un identificador hace referencia a una entidad como máximo.

2. Cada entidad es referida por al menos un identificador.

3. Un identificador siempre hace referencia a la misma entidad (es decir, nunca se reutiliza).

- (●) V
- ( ) F

La sincronización de transmisión de referencias (RBS, por sus siglas en inglés) es un protocolo de sincronización de relojes que difiere mucho de otras propuestas. Primero, el protocolo no asume que hay un solo nodo con una cuenta exacta del tiempo real disponible. En lugar de ayudar a proporcionar a todos los nodos el tiempo UTC, este protocolo ayuda simplemente a sincronizar internamente los relojes, justo como el algoritmo de Berkeley. Segundo, las soluciones explicadas hasta el momento están diseñadas para lograr que el emisor y el receptor estén sincronizados, esencialmente siguiendo un protocolo de dos vías...

* 5 puntos

- (●) V
- ( ) F