Eager y sus colaboradores (1986) construyeron un modelo de puesta en cola analítico de este algoritmo. Mediante el uso de este modelo se estableció que el algoritmo tiene un buen comportamiento y es estable bajo un amplio rango de parámetros, incluyendo diversos valores de umbral, costos de transferencia y límites de sondeo.

Sin embargo, hay que tener en cuenta que bajo condiciones de mucha carga, todas las máquinas enviarán sondas constantemente a otras máquinas, en un intento inútil por encontrar una que esté dispuesta a aceptar más trabajo. Se descargarán pocos procesos hacia la otra máquina, además de que se puede producir una considerable sobrecarga al tratar de hacerlo.

Un algoritmo heurístico distribuido, iniciado por el receptor

Hay un algoritmo complementario para el que analizamos en la sección anterior (que lo inicia un emisor sobrecargado), el cual lo inicia un receptor con poca carga, como se muestra en la figura 8-25(b). Con este algoritmo, cada vez que un proceso termina, el sistema comprueba si tiene suficiente trabajo. De no ser así, selecciona una máquina al azar y le pide trabajo. Si esa máquina no tiene nada que ofrecer, selecciona una segunda máquina y después una tercera. Si no se encuentra trabajo después de N sondeos, el nodo deja de pedir trabajo temporalmente, realiza cualquier trabajo que se haya puesto en cola e intenta de nuevo cuando termina el siguiente proceso. Si no hay trabajo disponible, la máquina queda inactiva. Después de cierto intervalo fijo, empieza a sondear de nuevo.

Una ventaja de este algoritmo es que no impone una carga adicional sobre el sistema en momentos críticos. El algoritmo iniciado por el emisor realiza muchos sondeos precisamente cuando el sistema tiene menos capacidad de tolerarlos: cuando tiene mucha carga. Con el algoritmo iniciado por el receptor, cuando el sistema tiene mucha carga hay muy poca probabilidad de que una máquina no tenga suficiente trabajo. Sin embargo, cuando ocurra esto será fácil encontrar trabajo por hacer. Desde luego que cuando hay poco trabajo por hacer, el algoritmo iniciado por el receptor crea un tráfico de sondeo considerable, ya que todas las máquinas desempleadas están buscando trabajo con desesperación. No obstante, es mucho mejor que aumente la sobrecarga cuando el sistema tiene poca carga que cuando está sobrecargado.

También es posible combinar ambos algoritmos, para que las máquinas traten de deshacerse de trabajo cuando tengan mucho, y traten de adquirir trabajo cuando no tengan suficiente. Además, las máquinas tal vez puedan mejorar el sondeo al azar si mantienen un historial de sondeos anteriores, para determinar si tienen un problema crónico de poca carga o de mucha carga. Se puede probar uno de estos algoritmos primero, dependiendo de si el iniciador está tratando de deshacerse de trabajo, o de adquirir más.

8.3 VIRTUALIZACIÓN

En ciertas situaciones, una empresa tiene una multicomputadora pero en realidad no la quiere. Un ejemplo común es cuando una empresa tiene un servidor de correo electrónico, un servidor Web, un servidor FTP, algunos servidores de comercio electrónico, y otros servidores más. Todos estos servidores se ejecutan en distintas computadoras del mismo bastidor de equipos, y todos están conec-

tados por una red de alta velocidad; en otras palabras, es una multicomputadora. En algunos casos, todos estos servidores se ejecutan en máquinas separadas debido a que una sola máquina no puede manejar la carga, pero en muchos otros casos la razón principal para no ejecutar todos estos servicios como procesos en la misma máquina es la confiabilidad: la administración simplemente no confía en que el sistema operativo se ejecute las 24 horas del día, los 365 o 366 días del año sin fallas. Al colocar cada servicio en una computadora separada, si falla uno de los servidores por lo menos los demás no se verán afectados. Aunque de esta forma se logra la tolerancia a fallas, esta solución es costosa y difícil de administrar debido a que hay muchas máquinas involucradas.

¿Qué se debe hacer? Se ha propuesto como solución la tecnología de las máquinas virtuales, que a menudo se conoce sólo como **virtualización** y tiene más de 40 años, como vimos en la sección 1.7.5. Esta tecnología permite que una sola computadora contenga varias máquinas virtuales, cada una de las cuales puede llegar a ejecutar un sistema operativo distinto. La ventaja de este método es que una falla en una máquina virtual no ocasiona que las demás fallen de manera automática. En un sistema virtualizado, se pueden ejecutar distintos servidores en diferentes máquinas virtuales, con lo cual se mantiene el modelo parcial de fallas que tiene una computadora, pero a un costo mucho menor y con una administración más sencilla.

Claro que consolidar los servidores de esta forma es como poner todos los huevos en una canasta. Si falla el servidor que ejecuta todas las máquinas virtuales, el resultado es aún más catastrófico que cuando falla un solo servidor dedicado. Sin embargo, la razón por la que la virtualización puede funcionar es que la mayoría de las fallas en los servicios no se deben a un hardware defectuoso, sino al software presuntuoso, poco confiable y lleno de errores, en especial los sistemas operativos. Con la tecnología de máquinas virtuales, el único software que se ejecuta en el modo del kernel es el hipervisor, el cual tiene 100 veces menos líneas de código que un sistema operativo completo, y por ende tiene 100 veces menos errores.

La ejecución de software en las máquinas virtuales tiene otras ventajas además de un sólido aislamiento. Una de ellas es que al tener menos máquinas físicas hay un ahorro en hardware y electricidad, y se ocupa menos espacio en la oficina. Para una compañía como Amazon, Yahoo, Microsoft o Google, que puede tener cientos de miles de servidores que realizan una enorme variedad de tareas distintas, la reducción de las demandas físicas en sus centros de datos representa un enorme ahorro en los costos. Por lo general, en las empresas grandes los departamentos individuales o grupos piensan en una idea interesante y después van y compran un servidor para implementarla. Si la idea tiene éxito y se requieren cientos o miles de servidores, se expande el centro de datos corporativo. A menudo es difícil mover el software a las máquinas existentes, debido a que cada aplicación necesita con frecuencia una versión distinta del sistema operativo, sus propias bibliotecas, archivos de configuración y demás. Con las máquinas virtuales, cada aplicación puede tener su propio entorno.

Otra ventaja de las máquinas virtuales es que es mucho más fácil usar puntos de comprobación y migrar datos entre una máquina virtual y otra (por ejemplo, para balancear la carga entre varios servidores) que migrar procesos que se ejecutan en un sistema operativo normal. En este último caso, se mantiene una cantidad considerable de información crítica de estado sobre cada proceso en las tablas del sistema operativo, incluyendo la información relacionada con la apertura de archivos, las alarmas, los manejadores de señales y demás. Al migrar una máquina virtual, todo lo que hay que mover es la imagen de memoria, ya que todas las tablas del sistema operativo se mueven también.

Otro uso para las máquinas virtuales es para ejecutar aplicaciones heredadas en los sistemas operativos (o en versiones de los sistemas operativos) que ya no tienen soporte o que no funcionan en el hardware actual. Estas aplicaciones heredadas se pueden ejecutar al mismo tiempo y en el mismo hardware que las aplicaciones actuales. De hecho, la habilidad de ejecutar al mismo tiempo aplicaciones que utilizan distintos sistemas operativos es un gran argumento a favor de las máquinas virtuales.

El desarrollo de software es otro uso aún más importante de las máquinas virtuales. Un programador que quiera asegurarse que su software funcione en Windows 98, Windows 2000, Windows XP, Windows Vista, varias versiones de Linux, FreeBSD, OpenBSD, NetBSD y Mac OS X ya no tiene que conseguir una docena de computadoras e instalar distintos sistemas operativos en todas ellas. Lo único que tiene que hacer es crear una docena de máquinas virtuales en una sola computadora e instalar distintos sistemas operativos en cada máquina virtual. Desde luego que el programador podría haber particionado el disco duro para instalar un sistema operativo distinto en cada partición, pero este método es más dificultoso. En primer lugar, las PCs estándar sólo soportan cuatro particiones primarias de disco, sin importar qué tan grande sea. En segundo lugar, aunque se podría instalar un programa de multiarranque en el bloque de arranque, sería necesario reiniciar la computadora para trabajar en otro sistema operativo. Con las máquinas virtuales todos los sistemas operativos se pueden ejecutar al mismo tiempo, ya que en realidad sólo son procesos glorificados.

8.3.1 Requerimientos para la virtualización

Como vimos en el capítulo 1, hay dos métodos para la virtualización. En la figura 1-29(a) se muestra un tipo de hipervisor, denominado **hipervisor de tipo 1** (o **monitor de máquina virtual**). En realidad es el sistema operativo, ya que es el único programa que se ejecuta en modo del kernel. Su trabajo es soportar varias copias del hardware actual, conocidas como **máquinas virtuales**, de una manera similar a los procesos que soporta un sistema operativo normal. Por el contrario, un **hipervisor de tipo 2** [que se muestra en la figura 1-29(b)] es algo completamente distinto. Es sólo un programa de usuario que se ejecuta (por decir) en Windows o Linux e "interpreta" el conjunto de instrucciones de la máquina, el cual también crea una máquina virtual. Pusimos "interpreta" entre comillas debido a que por lo general, los trozos de código se procesan de cierta forma para después colocarlos en la caché y ejecutarlos directamente para mejorar el rendimiento, pero la interpretación completa funcionaría en principio, aunque con lentitud. El sistema operativo que se ejecuta encima del hipervisor en ambos casos se denomina **sistema operativo invitado**. En el caso de un hipervisor de tipo 2, el sistema operativo que se ejecuta en el hardware se denomina **sistema operativo anfitrión**.

Es importante tener en cuenta que en ambos casos las máquinas virtuales deben actuar justo de igual forma que el hardware real. Específicamente, debe ser posible iniciarlas como máquinas reales e instalar cualquier sistema operativo en ellas, justo como lo que se puede hacer con el hardware real. La tarea del hipervisor es brindar esta ilusión con eficiencia (sin ser un intérprete completo).

La razón de que haya dos tipos de hipervisores se debe a ciertos defectos en la arquitectura del Intel 386 que se acarrearon servilmente a las nuevas CPUs durante 20 años, para mantener la compatibilidad inversa. En síntesis, cada CPU con modo de kernel y modo de usuario tiene un conjun-

to de instrucciones que sólo se pueden ejecutar en modo de kernel, como las instrucciones que realizan operaciones de E/S, las que modifican las opciones de la MMU, etcétera. Popek y Goldberg (1974) desarrollaron un trabajo clásico sobre virtualización, en el cual a estas instrucciones les llamaron **instrucciones sensibles**. También hay un conjunto de instrucciones que producen una trampa (interrupción) si se ejecutan en modo de usuario. A estas instrucciones, Popek y Goldberg les llamaron **instrucciones privilegiadas**. En su artículo declararon por primera vez que una máquina se puede virtualizar sólo si las instrucciones sensibles son un subconjunto de las instrucciones privilegiadas. Para explicarlo en un lenguaje más simple, si usted trata de hacer algo en modo de usuario que no deba hacer en este modo, el hardware deberá producir una interrupción. A diferencia de la IBM/370, que tenía esta propiedad, el Intel 386 no la tiene. Se ignoraban muchas instrucciones sensibles del 386 si se ejecutaba en modo de usuario. Por ejemplo, la instrucción POPF sustituye el registro de banderas, que modifica el bit que habilita/deshabilita las interrupciones. En modo de usuario, este bit simplemente no se modifica. Como consecuencia, el 386 y sus sucesores no se podían virtualizar, por lo que no podían soportar un hipervisor de tipo 1.

En realidad la situación es un poco peor de lo que se muestra. Además de los problemas con las instrucciones que no se pueden atrapar en modo de usuario, hay instrucciones que pueden leer el estado sensible en modo de usuario sin producir una interrupción. Por ejemplo, en el Pentium un programa puede determinar si se está ejecutando en modo de usuario o en modo de kernel con sólo leer su selector del segmento de código. Un sistema operativo que hiciera esto y descubriera que se encuentra en modo de usuario podría tomar una decisión incorrecta con base en esta información.

Este problema se resolvió cuando Intel y AMD introdujeron la virtualización en sus CPUs, empezando en el 2005. En las CPUs Intel Core 2 se conoce como **VT** (**Tecnología de virtualización**); en las CPUs AMD Pacific se conoce como **SVM** (**Máquina virtual segura**). A continuación utilizaremos el término VT en un sentido genérico. Ambas tecnologías se inspiraron en el trabajo de la IBM VM/370, pero tienen unas cuantas diferencias. La idea básica es crear contenedores en los que se puedan ejecutar máquinas virtuales. Cuando se inicia un sistema operativo invitado en un contenedor, se sigue ejecutando ahí hasta que produce una excepción y se atrapa en el hipervisor, por ejemplo, mediante la ejecución de una instrucción de E/S. El conjunto de operaciones que se atrapan se controla mediante un mapa de bits de hardware establecido por el hipervisor. Con estas extensiones, es posible utilizar el clásico método de atrapar y emular de una máquina virtual.

8.3.2 Hipervisores de tipo 1

La capacidad de virtualización es una cuestión importante, por lo que la examinaremos con más detalle. En la figura 8-26 podemos ver un hipervisor de tipo 1 que soporta una máquina virtual. Al igual que todos los hipervisores de tipo 1, se ejecuta en modo de kernel. La máquina virtual se ejecuta como un proceso de usuario en modo de usuario y, como tal, no puede ejecutar instrucciones sensibles. La máquina virtual ejecuta un sistema operativo invitado que piensa que se encuentra en modo de kernel, aunque desde luego se encuentra en modo de usuario. A éste le llamaremos **modo de kernel virtual**. La máquina virtual también ejecuta procesos de usuario, los cuales creen que se encuentran en modo de usuario (y en realidad así es).

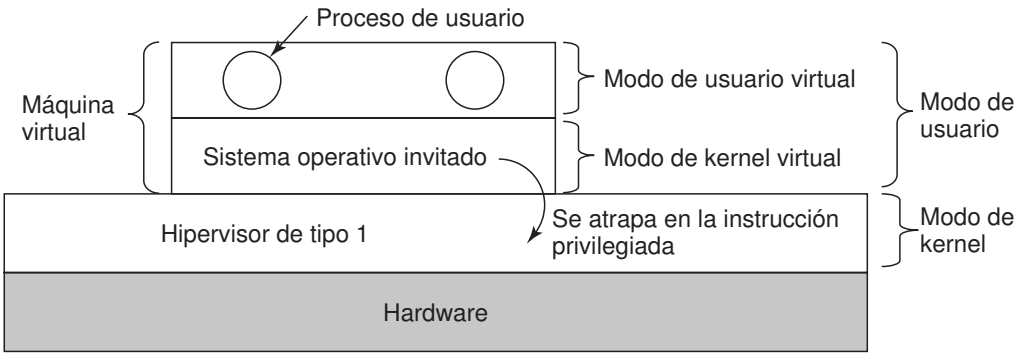

Figura 8-26. Cuando el sistema operativo en una máquina virtual ejecuta una instrucción que sólo se puede ejecutar en modo de kernel, se atrapa en el hipervisor si hay tecnología de virtualización presente.

¿Qué ocurre cuando el sistema operativo (el cual cree que se encuentra en modo de kernel) ejecuta una instrucción sensible (una que sólo se permite en modo de kernel)? En las CPUs sin VT, la instrucción falla y por lo general también lo hace el sistema operativo. Esto hace que la verdadera virtualización sea imposible. Sin duda podríamos argumentar que todas las instrucciones sensibles siempre se deben atrapar al ejecutarse en modo de usuario, pero esa no es la forma en que trabajaba el 386 y sus sucesores que no utilizaban la tecnología VT.

En las CPUs con VT, cuando el sistema operativo invitado ejecuta una instrucción sensible se produce una interrupción en el kernel, como se muestra en la figura 8-26. Así, el hipervisor puede inspeccionar la instrucción para ver si el sistema operativo anfitrión la emitió en la máquina virtual, o si fue un programa de usuario en la máquina virtual. En el primer caso, hace las preparaciones para que se ejecute la instrucción; en el segundo caso, emula lo que haría el hardware real al confrontarlo con una instrucción sensible que se ejecuta en modo de usuario. Si la máquina virtual no tiene VT, por lo general la instrucción se ignora; si tiene VT, se atrapa en el sistema operativo invitado que se ejecuta en la máquina virtual.

8.3.3 Hipervisores de tipo 2

El proceso de crear un sistema de máquina virtual es bastante simple cuando hay VT disponible, pero ¿qué hacían las personas antes de eso? Sin duda, no se podría ejecutar un sistema operativo completo en una máquina virtual debido a que sólo se ignorarían (algunas de) las instrucciones sensibles, y el sistema fallaría. En vez de ello, lo que ocurrió fue la invención de lo que ahora se conoce como **hipervisores de tipo 2**, como se muestra en la figura 1-29(b). El primero de estos hipervisores fue **VMware** (Adams y Agesen, 2006; y Waldspurger, 2002), el fruto del proyecto de investigación DISCO en la Universidad de Stanford (Bugnion y colaboradores, 1997). VMware se ejecuta como un programa de usuario ordinario encima de un sistema operativo anfitrión como Windows o Linux. Cuando se inicia por primera vez, actúa como una computadora que se acaba de iniciar y espera encontrar en la unidad de CD-ROM un disco que contenga un sistema operativo.

Después instala el sistema operativo en su **disco virtual** (que en realidad sólo es un archivo de Windows o Linux), para lo cual ejecuta el programa de instalación que se encuentra en el CD-ROM. Una vez que se instala el sistema operativo invitado en el disco virtual, se puede iniciar al ejecutarlo.

Ahora veamos cómo funciona VMware con un poco más de detalle. Al ejecutar un programa binario del Pentium, que puede obtener del CD-ROM de instalación o del disco virtual, primero explora el código para buscar **bloques básicos**; es decir, ejecuciones de instrucciones seguidas que terminan en un salto, una llamada, una interrupción o alguna instrucción que modifica el flujo de control. Por definición, ningún bloque básico contiene una instrucción que modifique el contador del programa, excepto el último. Se inspecciona el bloque básico para ver si contiene instrucciones sensibles (en el sentido de Popek y Goldberg). De ser así, cada una de estas instrucciones se sustituye con una llamada a un procedimiento de VMware que la maneja. La instrucción final también se sustituye con una llamada a VMware.

Una vez que se realicen estos pasos, el bloque básico se coloca en la caché de VMware y después se ejecuta. Un bloque básico que no contenga instrucciones sensibles se ejecutará con la misma rapidez bajo VMware que en la máquina básica (ya que se está ejecutando en esa máquina). Las instrucciones sensitivas se atrapan y emulan de esta manera. A esta técnica se le conoce como **traducción binaria**.

Una vez que el bloque básico haya completado su ejecución, el control se regresa a VMware, que localiza a su sucesor. Si ya se ha traducido el sucesor, se puede ejecutar de inmediato. Si no, primero se traduce, se coloca en la caché y después se ejecuta. En un momento dado, la mayor parte del programa estará en la caché y se ejecutará casi a la velocidad completa. Se utilizan varias optimizaciones; por ejemplo, si un bloque básico termina al saltar (o llamar) a otro, se puede sustituir la instrucción final mediante un salto o llamada directamente al bloque básico traducido, con lo cual se elimina toda la sobrecarga asociada con la búsqueda del bloque sucesor. Además, no hay necesidad de sustituir las instrucciones sensibles en los programas de usuario; el hardware las ignorará de todas formas.

Ahora debe estar claro por qué funcionan los hipervisores de tipo 2, incluso en hardware que no se puede virtualizar: todas las instrucciones sensibles se sustituyen mediante llamadas a procedimientos que emulan estas instrucciones. El verdadero hardware nunca ejecuta las instrucciones sensibles que emite el sistema operativo invitado. Se convierten en llamadas al hipervisor, quien a su vez las emula.

Tal vez podríamos esperar que las CPUs con VT tuvieran un rendimiento mucho mayor que las técnicas de software que se utilizan en los hipervisores de tipo 2, pero las mediciones muestran otra cosa (Adams y Agesen, 2006). Resulta ser que el método de atrapar y emular que utiliza el hardware con VT genera muchas interrupciones, y éstas son muy costosas en el hardware moderno, debido a que arruinan las cachés de las CPUs, los TLBs y las tablas de predicción de bifurcación internas para la CPU. Por el contrario, cuando las instrucciones sensibles se sustituyen mediante llamadas a procedimientos de VMware dentro del proceso que está en ejecución, no se produce sobrecarga debido a este cambio de contexto. Como muestran Adams y Agesen, algunas veces el software vence al hardware, dependiendo de la carga de trabajo. Por esta razón, algunos hipervisores de tipo 1 realizan la traducción binaria por cuestiones de rendimiento, aun cuando el software se ejecute de manera correcta sin ella.

8.3.4 Paravirtualización

Los hipervisores de tipo 1 y tipo 2 funcionan con sistemas operativos invitados que no estén modificados, pero tienen que hacer un gran esfuerzo por obtener un rendimiento razonable. Un método distinto que se está haciendo popular es modificar el código fuente del sistema operativo invitado, de manera que en vez de ejecutar instrucciones sensibles, realiza **llamadas al hipervisor**. En efecto, el sistema operativo invitado actúa como un programa de usuario que realiza llamadas al sistema operativo (el hipervisor). Cuando se utiliza este método, el hipervisor debe definir una interfaz que consiste en un conjunto de llamadas a procedimientos que los sistemas operativos invitados puedan utilizar. Este conjunto de llamadas forma en efecto una **API** (**Interfaz de programación de aplicaciones**), aun cuando es una interfaz para que la utilicen los sistemas operativos invitados, y no los programas de aplicación.

Si vamos un paso más allá, al eliminar todas las instrucciones sensibles del sistema operativo para que sólo haga llamadas al hipervisor para obtener servicios del sistema como la E/S, hemos convertido al hipervisor en un microkernel como el de la figura 1-26. Se dice que un sistema operativo invitado para el que se han eliminado de manera intencional (algunas) instrucciones sensibles está **paravirtualizado** (Barham y colaboradores, 2003; Whitaker y colaboradores, 2002). La emulación de instrucciones peculiares del hardware es una tarea desagradable y que consume mucho tiempo. Requiere una llamada al hipervisor, para después emular la semántica exacta de una instrucción complicada. Es mucho mejor tan sólo hacer que el sistema operativo invitado llame al hipervisor (o microkernel) para realizar las operaciones de E/S, y así en lo sucesivo. La principal razón por la que los primeros hipervisores sólo emulaban la máquina completa era por la falta de disponibilidad de código fuente para el sistema operativo invitado (por ejemplo, Windows), o debido a la gran cantidad de variantes (como Linux). Tal vez en el futuro se estandarizará la API del hipervisor/microkernel y los próximos sistemas operativos se diseñarán para llamar a esa API, en vez de utilizar instrucciones sensibles. Esto facilitaría el soporte y uso de la tecnología de las máquinas virtuales.

En la figura 8-27 se ilustra la diferencia entre la verdadera virtualización y la paravirtualización. Aquí tenemos dos máquinas virtuales que se soportan mediante hardware de VT. A la izquierda hay una versión sin modificar de Windows como el sistema operativo invitado. Cuando se ejecuta una instrucción sensible, el hardware hace que se atrape en el hipervisor, que a su vez la emula y regresa. A la derecha hay una versión de Linux modificada de manera que ya no contiene instrucciones sensibles. Así, cuando necesita realizar operaciones de E/S o modificar registros internos críticos (como el que apunta a las tablas de páginas), realiza una llamada al hipervisor para realizar el trabajo, de igual forma que cuando un programa de aplicación hace una llamada al sistema en Linux estándar.

En la figura 8-27 hemos mostrado que el hipervisor se divide en dos partes separadas por una línea punteada. En realidad sólo hay un programa que se ejecuta en el hardware. Una parte de este programa es responsable de interpretar las instrucciones sensibles que se atrapan, que en este caso son de Windows. La otra parte del programa sólo lleva a cabo las llamadas al hipervisor. En la figura, la segunda parte del programa se etiqueta como "microkernel". Si el hipervisor sólo ejecuta sistemas operativos invitados paravirtualizados, no hay necesidad de emular instrucciones sensibles y tenemos un verdadero microkernel que sólo proporciona servicios muy básicos, como despachar

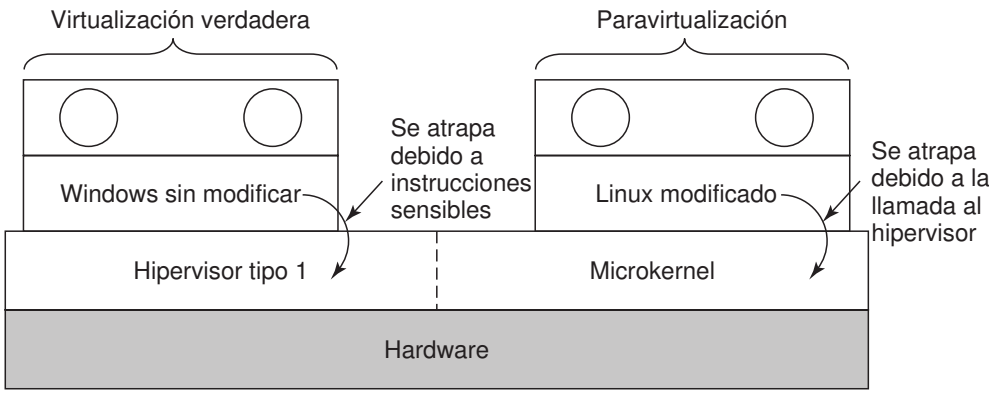

Figura 8-27. Un hipervisor que soporta la virtualización verdadera y la paravirtualización.

procesos y administrar la MMU. El límite entre un hipervisor de tipo 1 y un microkernel es de por sí impreciso, y empeorará aún más a medida que los hipervisores adquieran cada vez más funcionalidad y llamadas, según parece. Este tema es controversial, pero cada vez es más claro que el programa que se ejecute en modo de kernel en el hardware básico debe ser pequeño y confiable, y debe consistir de miles de líneas de código, no de millones de líneas. Varios investigadores han hablado sobre este tema (Hand y colaboradores, 2005; Heiser y colaboradores, 2006; Hohmuth y colaboradores, 2004; Roscoe y colaboradores, 2007).

Al paravirtualizar el sistema operativo invitado, surgen varias cuestiones. En primer lugar, si las instrucciones sensibles se sustituyen con llamadas al hipervisor, ¿cómo se puede ejecutar el sistema operativo en el hardware nativo? Después de todo, el hardware no comprende estas llamadas al hipervisor. Y en segundo lugar, ¿qué pasa si hay varios hipervisores disponibles en el mercado como VMware, el Xen de código fuente abierto (diseñado originalmente por la Universidad de Cambridge) y Microsoft Viridian, todos ellos con APIs de hipervisor algo distintas? ¿Cómo se puede modificar el kernel para ejecutarse en todos ellos?

Amsden y sus colaboradores (2006) han propuesto una solución. En su modelo, el kernel se modifica para llamar a ciertos procedimientos especiales cada vez que necesita hacer algo sensible. En conjunto, estos procedimientos conocidos como **VMI** (*Virtual Machina Interface,* Interfaz de máquina virtual) forman una capa de bajo nivel que actúa como interfaz para el hardware o hipervisor. Estos procedimientos están diseñados para ser genéricos y no estar enlazados al hardware o a un hipervisor en especial.

En la figura 8-28 se muestra un ejemplo de esta técnica para una versión paravirtualizada de Linux, conocida como VMI Linux (VMIL). Cuando VMI Linux se ejecuta en el hardware básico, tiene que vincularse con una biblioteca que emite la instrucción actual (sensible) necesaria para realizar el trabajo, como se muestra en la figura 8.28(a). Al ejecutarse en un hipervisor (por ejemplo, en VMware o Xen), el sistema operativo invitado se vincula con distintas bibliotecas que componen las llamadas apropiadas (y distintas) al hipervisor para el hipervisor subyacente. De esta forma, el núcleo del sistema operativo sigue siendo portable, amigable para los hipervisores y eficiente.

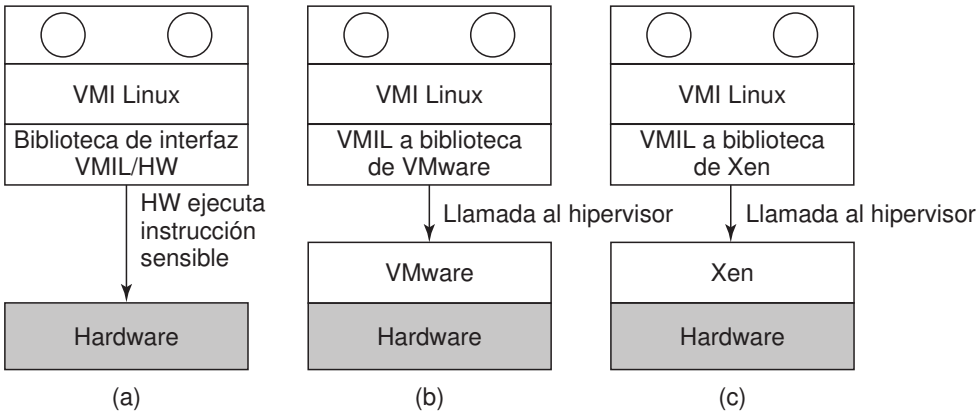

Figura 8-28. VMI Linux ejecutándose en (a) el hardware básico (b) VMware (c) Xen.

También se han propuesto otras técnicas para la interfaz de la máquina virtual. Una de las más populares es **paravirt ops**. Esta idea es similar en concepto a lo que hemos visto antes, pero difiere en algunos detalles.

8.3.5 Virtualización de la memoria

Hasta ahora sólo hemos tratado con la cuestión sobre cómo virtualizar la CPU. Pero un sistema de cómputo tiene más componentes aparte de la CPU. También tiene memoria y dispositivos de E/S. Éstos también se tienen que virtualizar. Veamos ahora cómo se hace esto.

Casi todos los sistemas operativos modernos soportan la memoria virtual, que en esencia es una asignación de las páginas en el espacio de direcciones virtuales a las páginas de la memoria física. Esta asignación se define mediante tablas de páginas (multinivel). Por lo general la asignación se establece en movimiento, al hacer que el sistema operativo establezca un registro de control en la CPU que apunte a la tabla de páginas de nivel superior. La virtualización complica de manera considerable la administración de la memoria.

Por ejemplo, suponga que se está ejecutando una máquina virtual y que el sistema operativo invitado en ella decide asignar sus páginas virtuales 7, 4 y 3 a las páginas físicas 10, 11 y 12, respectivamente. Crea tablas de páginas que contienen esta asignación y carga un registro de hardware para que apunte a la página de tablas de nivel superior. Esta instrucción es sensible. En una CPU con VT, se producirá una interrupción; con VMware producirá una llamada a un procedimiento de VMware; en un sistema operativo paravirtualizado generará una llamada al hipervisor. En aras de la simplicidad, vamos a suponer que se atrapa en un hipervisor de tipo 1, pero el problema es el mismo en los tres casos. ¿Qué hace ahora el hipervisor? Una solución es asignar las páginas físicas 10, 11 y 12 a esta máquina virtual y establecer las páginas de tablas actuales para que asignen las páginas virtuales 7, 4 y 3 de la máquina virtual para utilizarlas. Hasta ahora todo va bien.

Ahora suponga que se inicia una segunda máquina virtual y que asigna sus páginas virtuales 4, 5 y 6 a las páginas físicas 10, 11 y 12, y que carga el registro de control para que apunte a sus tablas de páginas. El hipervisor atrapa la interrupción, pero ¿qué debe hacer? No puede utilizar esta asignación debido a que las páginas físicas 10, 11 y 12 ya están en uso. Puede buscar páginas libres, por ejemplo 20, 21 y 22, y utilizarlas, pero primero tiene que crear nuevas páginas de tablas para asignar los páginas virtuales 4, 5 y 6 de la máquina virtual 2 a las páginas físicas 20, 21 y 22. Si se inicia otra máquina virtual y trata de utilizar las páginas físicas 10, 11 y 12, tiene que crear una asignación para ella. En general, para cada máquina virtual el hipervisor necesita crear una **tabla de páginas oculta** (**shadow**) que asigne las páginas virtuales que utiliza la máquina virtual a las páginas actuales que el hipervisor le otorgó.

Lo que es peor aún, cada vez que el sistema operativo invitado cambie sus tablas de páginas, el hipervisor también tiene que cambiar las tablas de páginas ocultas. Por ejemplo, si el SO invitado reasigna la página virtual 7 a lo que ve como la página virtual 200 (en vez de 10), el hipervisor tiene que saber acerca de este cambio. El problema es que el sistema operativo invitado puede cambiar sus tablas de páginas con sólo escribir en la memoria. No se requieren operaciones sensibles, por lo que el hipervisor ni siquiera se entera sobre el cambio y, en definitiva, no puede actualizar las tablas de páginas ocultas que utiliza el hardware actual.

Una posible solución (pero burda) es que el hipervisor lleve el registro sobre cuál página en la memoria virtual del invitado contiene la tabla de páginas de nivel superior. Puede obtener esta información la primera vez que el invitado intente cargar el registro de software al que apunta, ya que esta instrucción es sensible y produce una interrupción. El hipervisor puede crear una tabla de páginas oculta en este punto, y también puede asignar la tabla de páginas de nivel superior y las tablas de páginas a las que apunta para que sean de sólo lectura. Los intentos subsiguientes del sistema operativo invitado por modificar cualquiera de estas páginas producirán un fallo de página y, en consecuencia, otorgarán el control al hipervisor, que puede analizar el flujo de instrucciones, averiguar qué es lo que está tratando de hacer el SO invitado y actualizar las tablas de páginas ocultas según sea necesario. No es bonito, pero se puede hacer en principio.

Ésta es un área en la que las futuras versiones de VT podrían ofrecer asistencia al realizar una asignación de dos niveles en el hardware. Primero, el hardware podría asignar la página virtual a la idea que tiene el invitado sobre la página física, y después podría asignar esa dirección (que el hardware ve como una dirección virtual) a la dirección física, todo sin que se produzcan interrupciones. De esta forma no habría que marcar ninguna tabla de páginas como de sólo lectura, y el hipervisor simplemente tendría que proveer una asignación entre el espacio de direcciones virtuales de cada invitado y la memoria física. Al cambiar de máquinas virtuales, sólo cambiaría esta asignación de la misma forma en que un sistema operativo normal cambia la asignación al cambiar de procesos.

En un sistema operativo paravirtualizado, la situación es diferente. Aquí, el SO paravirtualizado en el invitado sabe que cuando termine de cambiar la tabla de páginas de cierto proceso, debe informar al hipervisor. En consecuencia, primero cambia la tabla de páginas por completo y después emite una llamada al hipervisor para indicarle sobre la nueva tabla de páginas. Así, en vez de obtener un fallo de protección en cada actualización a la tabla de páginas, hay una llamada al hipervisor cuando se han realizado todas las actualizaciones. Ésta es una manera más eficiente de hacer las cosas.

8.3.6 Virtualización de la E/S

Ya que analizamos la virtualización de la CPU y de la memoria, el siguiente paso es examinar la virtualización de la E/S. Por lo general, el sistema operativo invitado empezará con un sondeo del hardware para averiguar qué tipos de dispositivos de E/S están conectados. Estas sondas producirán interrupciones en el hipervisor. ¿Qué debe hacer el hipervisor? Un método es que reporte de vuelta que los discos, las impresoras y demás dispositivos son los que realmente tiene el hardware. Después el invitado cargará drivers para estos dispositivos y tratará de utilizarlos. Cuando los drivers de los dispositivos traten de realizar operaciones de E/S, leerán y escribirán en los registros de dispositivo del hardware del dispositivo correspondiente. Estas instrucciones son sensibles y se atraparán en el hipervisor, que podría entonces copiar los valores necesarios que entran y salen de los registros de hardware, según sea necesario.

Pero aquí también tenemos un problema. Cada SO invitado piensa que posee toda una partición de disco, y puede haber muchas máquinas virtuales más (cientos) que particiones de disco. La solución usual es que el hipervisor cree un archivo o región en el disco para el disco físico de cada máquina virtual. Como el SO invitado está tratando de controlar un disco que el hardware real tiene (y que el hipervisor conoce), puede convertir el número de bloque al que se está accediendo en un desplazamiento en el archivo o región que se está utilizando para el almacenamiento, y así puede realizar la operación de E/S.

También es posible que el disco que utiliza el invitado sea diferente del disco real. Por ejemplo, si el disco actual es un disco nuevo de alto rendimiento (o RAID) con una nueva interfaz, el hipervisor podría anunciar al SO invitado que tiene un simple disco IDE antiguo, para permitirle que instale un driver de disco IDE. Cuando este driver emite los comandos de disco IDE, el hipervisor los convierte en comandos para controlar el nuevo disco. Esta estrategia se puede utilizar para actualizar el hardware sin tener que cambiar el software. De hecho, esta habilidad de las máquinas virtuales para reasignar dispositivos de hardware fue una de las razones por las que la VM/370 se hizo popular: las empresas querían comprar hardware nuevo y más veloz, pero no querían cambiar su software. Esto fue posible gracias a la tecnología de las máquinas virtuales.

Otro problema de E/S que se debe resolver de alguna manera es el uso de DMA, que utiliza direcciones de memoria absolutas. Como podría esperarse, el hipervisor tiene que intervenir aquí y reasignar las direcciones antes de que inicie el DMA. Sin embargo, está empezando a aparecer el hardware con una **MMU de E/S**, que virtualiza la E/S de la misma forma en que la MMU virtualiza la memoria. Este hardware elimina el problema del DMA.

Un método distinto para manejar la E/S es dedicar una de las máquinas virtuales para que ejecute un sistema operativo estándar y que le refleje todas las llamadas a la E/S de las demás máquinas virtuales. Este método se mejora cuando se utiliza la paravirtualización, por lo que el comando que se emite al hipervisor en realidad indica lo que el SO invitado desea (por ejemplo, leer el bloque 1403 del disco 1), en vez de que sea una serie de comandos para escribir a los registros de dispositivo, en cuyo caso el hipervisor tiene que jugar a ser Sherlock Holmes para averiguar qué es lo que está tratando de hacer. Xen utiliza este método para la E/S, y la máquina virtual que se encarga de la E/S se llama **dominio 0**.

La virtualización de la E/S es un área en la que los hipervisores de tipo 2 tienen una ventaja práctica sobre los hipervisores de tipo 1: el sistema operativo anfitrión contiene los drivers de dis-

positivos para todos los extraños y maravillosos dispositivos de E/S conectados a la computadora. Cuando un programa de aplicación intenta acceder a un dispositivo E/S extraño, el código traducido puede llamar al driver de dispositivo existente para que realice el trabajo. Con un hipervisor de tipo 1, el hipervisor debe contener el driver o debe realizar una llamada a un driver en el dominio 0, que es algo similar a un sistema operativo anfitrión. A medida que madure la tecnología de las máquinas virtuales, es probable que el futuro hardware permita que los programas de aplicación accedan al hardware directamente de una manera segura, lo cual significa que los drivers de dispositivos podrán vincularse directamente con el código de la aplicación, o se podrán colocar en servidores separados en modo de usuario, con lo cual se eliminará el problema.

8.3.7 Dispositivos virtuales

Las máquinas virtuales ofrecen una interesante solución a un problema que desde hace mucho tiempo perturba a los usuarios, en especial a los del software de código fuente abierto: cómo instalar nuevos programas de aplicación. El problema es que muchas aplicaciones dependen de otras tantas aplicaciones y bibliotecas, que a su vez dependen de muchos otros paquetes de software, y así en lo sucesivo. Lo que es más, puede haber dependencias en versiones específicas de los compiladores, lenguajes de secuencias de comandos y en el sistema operativo.

Ahora que están disponibles las máquinas virtuales, un desarrollador de software puede construir con cuidado una máquina virtual, cargarla con el sistema operativo, compiladores, bibliotecas y código de aplicación requeridos, y congelar la unidad completa, lista para ejecutarse. Esta imagen de la máquina virtual se puede colocar en un CD-ROM o en un sitio Web para que los clientes la instalen o descarguen. Este método implica que sólo el desarrollador de software tiene que conocer todas las dependencias. Los clientes reciben un paquete completo que funciona de verdad, sin importar qué sistema operativo estén ejecutando ni qué otro software, paquetes y bibliotecas tengan instalados. A estas máquinas virtuales "empaquetadas y listas para usarse" se les conoce comúnmente como **dispositivos virtuales**.

8.3.8 Máquinas virtuales en CPUs de multinúcleo

La combinación de las máquinas virtuales y las CPUs de multinúcleo abre todo un nuevo mundo, en donde el número de CPUs disponibles se puede establecer en el software. Por ejemplo, si hay cuatro núcleos y cada uno de ellos se puede utilizar para ejecutar, digamos, hasta ocho máquinas virtuales, se puede configurar una sola CPU (escritorio) como una multicomputadora de 32 nodos si es necesario, pero también puede tener menos CPUs dependiendo de las necesidades del software. Nunca antes había sido posible que un diseñador de aplicaciones seleccionara primero cuántas CPUs quiere, para después escribir el software de manera acorde. Sin duda, esto representa una nueva fase en la computación.

Aunque todavía no es tan común, sin duda es concebible que las máquinas virtuales pudieran compartir la memoria. Todo lo que hay que hacer es asignar las páginas físicas en los espacios

direcciones de varias máquinas virtuales. Si se puede hacer esto, una sola computadora se puede convertir en un multiprocesador virtual. Como todos los núcleos en un chip multinúcleo comparten la misma RAM, un solo chip con cuádruple núcleo se podría configurar fácilmente como un multiprocesador de 32 nodos o como una multicomputadora de 32 nodos, según sea necesario.

La combinación de multinúcleo, máquinas virtuales, hipervisores y microkernels va a cambiar de manera radical la forma en que las personas piensan sobre los sistemas de cómputo. El software actual no puede lidiar con la idea de que el programador determine cuántas CPUs se necesitan, si se deben establecer como una multicomputadora o como un multiprocesador, y qué papel desempeñan los kernels mínimos de un tipo u otro. El software futuro tendrá que lidiar con estas cuestiones.

8.3.9 Cuestiones sobre licencias

Las licencias de la mayoría del software son por cada CPU. En otras palabras, cuando usted compra un programa, tiene el derecho de ejecutarlo sólo en una CPU. ¿Acaso este contrato le otorga el derecho de ejecutar el software en varias máquinas virtuales, todas las cuales se ejecutan en la misma máquina? Muchos distribuidores de software están algo inseguros sobre lo que deben hacer aquí.

El problema es mucho peor en las empresas que tienen una licencia que les permite tener *n* máquinas ejecutando el software al mismo tiempo, en especial cuando aumenta y disminuye la demanda de las máquinas virtuales.

En algunos casos, los distribuidores de software han colocado una cláusula explícita en la licencia, en la que se prohíbe al concesionario ejecutar el software en una máquina virtual o en una máquina virtual no autorizada. Está por verse si alguna de estas restricciones será válida en una corte y la manera en que los usuarios responderán a ellas.

8.4 SISTEMAS DISTRIBUIDOS

Ahora que hemos completado nuestro estudio de los multiprocesadores, las multicomputadoras y las máquinas virtuales, es tiempo de analizar el último tipo de sistema de múltiples procesadores: el **sistema distribuido**. Estos sistemas son similares a las multicomputadoras en cuanto a que cada nodo tiene su propia memoria privada, sin memoria física compartida en el sistema. Sin embargo, los sistemas distribuidos tienen un acoplamiento aún más débil que las multicomputadoras.

Para empezar, los nodos de una multicomputadora comúnmente tienen una CPU, RAM, una interfaz de red y tal vez un disco duro para la paginación. Por el contrario, cada nodo en un sistema distribuido es una computadora completa, con todo un complemento de periféricos. Además, los nodos de una multicomputadora por lo general están en un solo cuarto, de manera que se pueden comunicar mediante una red dedicada de alta velocidad, mientras que los nodos de un sistema distribuido pueden estar esparcidos en todo el mundo. Por último, todos los nodos de una computadora ejecutan el mismo sistema operativo, comparten un solo sistema de archivos y están bajo una administración común, mientras que cada uno de los nodos de un sistema distribuido puede ejecutar un sistema operativo diferente, cada uno con su propio sistema de archivos, y pueden estar