Virtualización de la memoria

El sistema de memoria es un componente fundamental de un computador y, por tanto, su virtualización eficiente es un aspecto clave a la hora de implementar un hipervisor. Para entender la dificultad de esta labor, hay que tener en cuenta que toda dirección generada en una máquina virtual (tanto si corresponde a una instrucción como a un dato; o tanto sea una dirección de usuario como de sistema) tiene que ser virtualizada, es decir, convertida a la dirección física correspondiente de la memoria real donde está alojada la máquina virtual. Nótese la diferencia con la virtualización del procesador, en la que sólo es necesario virtualizar las instrucciones conflictivas, ejecutando las restantes sin ningún tipo de intervención del hipervisor. La virtualización de la memoria, por tanto, requiere la intervención continua del hardware de gestión de memoria para poder llevarse a cabo de forma eficiente; pero ese hardware, al menos hasta hace poco tiempo, es ajeno a esta labor y hay que engañarle de alguna forma para conseguir el modo de operación requerido.

De manera similar a lo que ocurre con la virtualización del procesador y su vinculación con el gestor de procesos del sistema operativo, la funcionalidad requerida para virtualizar la memoria está muy relacionada con la asociada al propio gestor de memoria del sistema operativo (después de todo, el gestor de memoria proporciona memoria virtual y el hipervisor debe virtualizar la memoria virtual). Esta similitud no es sorprendente dado que una de las labores principales del sistema operativo es crear la abstracción de proceso que, al fin y al cabo, es una especie de máquina virtual extendida que ofrece a un programa en ejecución la impresión de que tiene un procesador y una memoria sólo para él. Por tanto, de la misma manera que muchos de los conceptos vinculados con los algoritmos de planificación de procesos de un sistema operativo pueden aplicarse, con algunas adaptaciones y extensiones, a la virtualización del procesador por parte del hipervisor (por ejemplo, el algoritmo de planificación de procesos CFS de Linux también lo puede usar, con ciertas extensiones, un hipervisor para planificar máquinas virtuales), las técnicas usadas por el gestor de memoria del sistema operativo pueden trasladarse y adaptarse a la virtualización de la memoria que lleva a cabo el hipervisor:

- De la misma manera que el gestor de memoria del sistema operativo ofrece un espacio de memoria lógico independiente y contiguo (con direcciones desde 0 hasta un cierto valor *N*) a cada proceso, el hipervisor debe proporcionar a cada máquina virtual un espacio de memoria físico (presuntamente, físico) independiente y contiguo (con direcciones desde 0 hasta un cierto valor *N*). Para entender la necesidad de esta contigüidad en el mapa físico, hay que tener en cuenta que algunos sistemas operativos requieren una memoria física contigua que comience por cero para poder funcionar correctamente. En cualquier caso, en este aspecto, hay algunas diferencias significativas entre estos dos niveles de funcionalidad: a cada proceso se le asigna un mapa lógico lo más grande posible extendiéndose por todo el rango de direcciones lógicas que ofrece el procesador, mientras que a una máquina virtual se le asigna un mapa físico del tamaño especificado por el usuario que crea esa máquina.
- De igual manera que el sistema operativo gestiona estructuras de datos que, por un lado, mantienen la correspondencia entre las páginas de cada proceso y los marcos de memoria asignados a las mismas (las tablas de páginas de cada proceso gestionadas por el sistema operativo) y, por otro lado, una estructura de datos que identifica qué marcos están libres (la tabla de marcos), el hipervisor debe gestionar estructuras de datos funcionalmente equivalentes: las tablas que hagan corresponder los marcos de cada máquina virtual con los marcos de la memoria real asignados a los mismos y una tabla que mantenga el estado de ocupación de todos los marcos de la memoria real.
- Igual que el gestor de memoria del sistema operativo reparte la memoria física entre los procesos mediante políticas de asignación de espacio y de reemplazo que intentan que los conjuntos de trabajo de los procesos estén residentes en memoria y usa un dispositivo de memoria secundaria (denominado, habitualmente, dispositivo de paginación o de *swap*) para almacenar las páginas que no caben en la memoria principal, el hipervisor debe repartir la memoria física real entre las máquinas virtuales teniendo en cuenta el comportamiento de cada máquina virtual, así como parámetros de configuración especificados por el usuario cuando crea la misma (como, por ejemplo, cantidad mínima y máxima de memoria física real asignada a una máquina virtual o su importancia o peso relativo con respecto a las otras máquinas virtuales que gestiona ese hipervisor para tenerlo en cuenta a la hora de realizar el reparto de la memoria entre las distintas máquinas virtuales), utilizando también un dispositivo de paginación para almacenar las páginas de las máquinas virtuales

que no caben en la memoria física real. A pesar de esas similitudes, en cuanto a este aspecto, hay diferencias significativas entre ambos niveles. Así, el algoritmo de reemplazo del hipervisor normalmente no dispondrá de información sobre cómo usan las páginas los procesos de cada sistema operativo lo que puede dificultar la adaptación directa de los algoritmos de reemplazo planteados en la literatura de sistemas operativos.

Recapitulando, a pesar de que la virtualización de la memoria por parte del hipervisor puede sacar provecho y reutilizar parte de la funcionalidad que implementa el gestor de memoria del sistema operativo, tiene que incidir en dos aspectos específicos: cómo manejar el hardware de gestión de memoria (basado, normalmente, en la técnica de la paginación) para conseguir que opere en un entorno de virtualización a pesar de no estar preparado para ello, y la adaptación, así como la concepción de nuevas técnicas cuando así se requiera, de los esquemas de reparto de memoria que usa el sistema operativo para aplicarlos a las máquinas virtuales. En este documento se analiza el primer aspecto (es decir, cómo virtualizar la paginación). Para realizar este análisis se usa como ejemplo la familia de procesadores x86 por su uso extendido, aunque, como bien es sabido, presentan serias dificultades para ser virtualizados. En el caso de la memoria, hay dos características de estos procesadores que no facilitan su virtualización: el uso de una TLB con gestión hardware (todo componente que se gestione por software es, obviamente, más fácil de virtualizar) que, además, no incluye identificadores de espacios de direcciones (ASID), lo que impide que pueda convivir en la TLB información de memoria asociada a distintas máquinas virtuales, siendo necesario invalidarla completamente cuando el hipervisor reasigna un procesador a otra máquina virtual (como ya es sabido, también es necesario invalidarla ante un cambio de proceso aunque ambos ejecuten en la misma máquina virtual). Este análisis se extenderá a las tres soluciones habituales para lograr la virtualización:

- Sistemas con paravirtualización, como, por ejemplo, Xen, en los que el sistema operativo alojado se modifica para facilitar una virtualización eficiente.
- Sistemas con virtualización completa sobre un procesador sin soporte para la misma, como, por ejemplo, VMware sobre un procesador x86 convencional, que representan el reto con mayor complejidad.
- Sistemas con virtualización completa sobre un procesador que da soporte a dicha técnica, como, por ejemplo, los procesadores x86 con las extensiones de virtualización de Intel (Intel VT-x) o AMD (AMD-V).

Virtualización de la paginación

En un sistema virtualizado se manejan tres tipos o niveles de direcciones de memoria:

- Direcciones lógicas (*DL*), que son las que generan los procesos que ejecutan en un sistema operativo alojado en una máquina virtual tanto cuando lo hacen en modo usuario, ejecutando el código de su programa, como cuando lo hacen en modo sistema, al llevar a cabo, por ejemplo, una llamada al sistema.
- Direcciones físicas virtuales (*DFV*), corresponden a las direcciones físicas que gestiona el sistema operativo alojado para el que, a todos los efectos, son direcciones reales de la memoria de la máquina.
- Direcciones físicas reales (*DFR*), que sólo son visibles para el hipervisor. Éstas son las únicas direcciones que pone el procesador en el bus de direcciones y que llegan a la memoria seleccionando la celda correspondiente.

Se precisan, por tanto, al menos conceptualmente, dos niveles de traducción:

- De direcciones lógicas a físicas virtuales (*DL→DFV*). Esta labor corresponde al sistema operativo. Para ello, como ya es conocido, el sistema operativo especifica por cada proceso una tabla de páginas que define la correspondencia entre el espacio lógico del proceso y el espacio físico asignado al mismo.
- De direcciones físicas virtuales a direcciones físicas reales (*DFV→DFR*). El hipervisor debe encargarse de gestionar este nivel de traducción puesto que es el componente que conoce qué memoria física real tiene asignada cada máquina virtual que gestiona. Para ello, usará por cada máquina virtual una estructura de datos que define la correspondencia entre el espacio físico virtual de esa máquina y el espacio físico real asignado a la misma.

Cuando se usa un procesador convencional, sin soporte para la virtualización, el hardware de gestión de memoria está diseñado para gestionar un único nivel de traducción: de las direcciones que generan los procesos (*DL*) a las direcciones que se envían a la memoria del sistema (*DFR*), implementando, por tanto, una traducción (*DL→DFR*). El problema está en que es el sistema operativo el que conoce acerca de las direcciones lógicas, pero es el hipervisor el que sabe acerca de las direcciones físicas reales. El reto que se presenta para conseguir una virtualización completa con este tipo de hardware es cómo logra el hipervisor manipular este hardware para implementar de forma compacta estos dos niveles de traducción requeridos. Para comprender cómo puede llevar a cabo su labor el hipervisor y dado que el trabajo principal de virtualización de la memoria, por motivos de rendimiento, lo lleva a cabo directamente el hardware de gestión de memoria, es conveniente analizar en qué puntos puede tomar control el hipervisor para realizar su trabajo de virtualización de la memoria, es decir, qué acciones del sistema operativo alojado o de los procesos que ejecutan en el mismo provocan la activación del hipervisor:

- Cuando se produce un fallo de página, al tratarse de una excepción, recibe el control de ejecución el hipervisor que, como se verá más adelante, dependiendo de las circunstancias que concurran, puede propagarlo al sistema operativo alojado o no.
- Cuando el sistema operativo alojado modifica el registro del procesador que apunta a las tablas de páginas (lo denominaremos *RBTP*: registro base a la tabla de páginas; en los procesadores x86, se trata del registro *CR3*) para que haga referencia a la tabla de páginas de un determinado proceso, al tratarse de una instrucción privilegiada, toma control el hipervisor. Nótese que esta acción la realiza el sistema operativo cada vez que hay un cambio de contexto entre dos procesos, ya que cada uno tiene su propia tabla de páginas. En el caso de los procesadores x86, al usar una TLB sin identificadores, esta operación produce automáticamente una invalidación del contenido actual de la TLB. Téngase en cuenta que, como ocurre con todos los registros del procesador, el hipervisor también debe virtualizar el *RBTP*, manteniendo una copia del mismo en la estructura donde almacena el estado de cada máquina virtual y restaurándolo al cambiar de máquina virtual (el cambio de máquina virtual podría considerarse como un cambio de contexto entre el proceso actual de la primera máquina virtual y el proceso actual de la segunda).
- En el momento que el sistema operativo alojado ejecuta una instrucción que solicita eliminar la entrada de la TLB que corresponda a una determinada dirección lógica (en los procesadores x86, se trata de la instrucción *INVLPG*), dado que es una instrucción privilegiada, toma control el hipervisor. Nótese que el hipervisor también tendrá que usar directamente esta instrucción cuando realiza algún cambio en la correspondencia entre direcciones físicas virtuales y direcciones físicas reales (por ejemplo, cuando expulsa una página asignada a una máquina virtual escribiéndola en el dispositivo de paginación del hipervisor).
- Hay que tener en cuenta que una parte muy importante de la gestión de memoria por parte del sistema operativo consiste en acceder y modificar entradas de tablas de páginas. Esta labor consiste meramente en accesos a memoria y, por tanto, el hipervisor no tiene control sobre la misma. En caso de que se requiera ese control por parte del hipervisor para implementar una determinada solución, éste tendrá que impedir al sistema operativo el acceso a las tablas de páginas modificando los permisos de acceso a las mismas de manera que se produzca una excepción cuando el sistema operativo alojado intente manipularlas.

Para facilitar el estudio de algunas de las soluciones al problema de la virtualización de la memoria que se va a realizar en el resto del documento, se va a repasar brevemente qué acciones típicas de gestión de memoria realiza un sistema operativo:

- Creación del mapa de memoria inicial de un proceso. El sistema operativo debe crear la tabla de páginas inicial del proceso. Nótese que el hipervisor no puede ser consciente de esta operación y no podrá descubrir esta tabla de páginas hasta que sea referenciada por el *RBTP* la primera vez que ejecute ese nuevo proceso.
- El proceso en ejecución causa un fallo de página: el sistema operativo busca un marco libre, lo rellena con el contenido de la página y escribe en la entrada correspondiente el número de ese marco.
- Durante la ejecución del proceso, el sistema operativo puede consultar y modificar una entrada de la tabla de páginas del mismo (algunos ejemplos que causarían estas operaciones serían: cambiar los

permisos de una página, poner a 0 el bit de referencia o de modificación de una página, o invalidar la entrada porque la página ha sido reemplazada o la región a la que pertenece la página ha sido eliminada). En caso de que se modifique la entrada, el sistema operativo tiene que ejecutar la instrucción privilegiada que invalida la entrada de la TLB correspondiente a esa página.

- Hay un cambio de contexto: el sistema operativo actualiza el valor del *RBTP* para que haga referencia a la tabla de páginas del proceso que entra a ejecutar. Esa operación causa la invalidación de la TLB.
- Nótese que, aunque un proceso no esté en ejecución, el sistema operativo puede tener que cambiar sus tablas de páginas. Por ejemplo, el algoritmo de reloj puede poner a 0 el bit de referencia de las páginas a las que se les da una segunda oportunidad o expulsar páginas de cualquier proceso.

A continuación, se presentan varios posibles enfoques a la hora de afrontar la virtualización de la memoria, tratando en el último punto el caso de procesadores que sí dan soporte a la virtualización de memoria.

- Manejo directo de las tablas de páginas del procesador.
- TLB virtual.
- Tablas de páginas en la sombra.
- Uso de procesadores con soporte para la virtualización.

Manejo directo de las tablas de páginas del procesador

Empecemos con una solución directa e intuitiva, pero tan ineficiente que no es factible su uso en un sistema real. Esta solución inicial plantea que el sistema operativo alojado, de la misma manera que ocurre en un sistema no virtualizado, trabaje directamente con el hardware de gestión de memoria del procesador construyendo una tabla de páginas para cada proceso e instalando en cada momento la tabla de páginas del proceso en ejecución, escribiendo para ello la dirección de la misma en el *RBTP*. En un cambio de proceso, el sistema operativo alojado simplemente asignaría a ese registro la dirección de la tabla de páginas del proceso elegido por el planificador del sistema operativo.

El problema de esta solución es que los números de marco de página que gestiona el sistema operativo alojado no corresponden, evidentemente, a los números de marco de la máquina real a los que están asignados. Así, por ejemplo, un hipotético sistema operativo alojado puede asignarle el marco 0 a una determinada página de un proceso y, por tanto, escribir un 0 en la entrada correspondiente a la tabla de páginas del mismo. Sin embargo, el hipervisor ha podido asignarle a ese marco 0 de la máquina virtual el marco 100 de la máquina física. Por tanto, en la entrada correspondiente de la tabla de páginas no habría que dejar que el sistema operativo escribiera un 0, sino que el hipervisor debería tomar control interceptando esa operación y escribiendo un 100 en su lugar. Pero no sólo es eso, sino que además, cuando el sistema operativo lea esa entrada, debe obtener un 0 y no el 100 que está realmente almacenado en la misma.

Para lograr ese comportamiento, el hipervisor debe prohibir el acceso a una tabla de páginas desde el momento que la descubre la primera vez que es referenciada desde el *RBTP*, de manera que toda lectura o escritura de la misma causará una excepción que tratará el hipervisor. El hipervisor usando sus estructuras de datos que relacionan los marcos de una máquina virtual con los marcos reales asignados a los mismos hará que las lecturas de la entrada por parte del sistema operativo obtengan el número de marco dentro de la memoria física de la máquina virtual, mientras que en las escrituras del sistema operativo alojado se sustituirá el número de marco que pretende escribir por el valor del marco real asignado por el hipervisor.

Dado el gran volumen de accesos que realiza el sistema operativo a las tablas de páginas de los procesos (pensemos, por ejemplo, en una llamada *fork* que tiene que leer toda la tabla de páginas de un proceso para duplicarla), se trataría de una solución no factible por su ineficiencia. En cualquier caso, aunque no sea una solución realista, para que sirva de comparativa con las soluciones que se presentan en los siguientes apartados, es interesante resaltar que con este esquema se requerirían las siguientes estructuras de datos para la gestión de memoria:

- Cada sistema operativo alojado gestionará una tabla de páginas para cada proceso, siendo estas tablas las que se instalan directamente en el hardware de gestión de memoria (en el *RBTP*).
- El hipervisor mantendrá por cada máquina virtual una tabla, no vinculada con el hardware de gestión de memoria, que mantendrá las correspondencias entre los marcos de esa máquina virtual y los marcos de la memoria real asignados a los mismos.

De todas formas, esta solución sí es factible en un entorno de paravirtualización, dado que en el mismo se puede modificar el sistema operativo alojado para lograr mayor eficiencia en la solución. Así, se puede añadir al sistema operativo la capacidad de gestionar la asignación de marcos reales a los marcos de la memoria de la máquina virtual (estamos haciendo que el sistema operativo alojado gestione los dos niveles de traducción), permitiendo así que el sistema operativo pueda consultar las entradas de las tablas de páginas sin ninguna sobrecarga. Sin embargo, hay que resaltar que, incluso en este caso, no se puede permitir que el sistema operativo escriba directamente sobre las entradas de las tablas de páginas puesto que eso le permitiría, ya sea por error o intencionadamente intentando romper la seguridad del sistema, hacer referencia a cualquier marco físico. Por tanto, la modificación de las entradas seguiría requiriendo la intervención del hipervisor para asegurarse de que la máquina virtual sólo accede a la memoria que le ha sido asignada.

TLB virtual

En esta solución, en ningún momento el hardware de gestión de memoria usa las tablas de páginas que construye el sistema operativo alojado (es decir, el *RBTP* nunca hace referencia a una tabla de páginas construida por el sistema operativo). El hipervisor (y, en consecuencia, el hardware de gestión de memoria) gestiona una única tabla de páginas por cada máquina virtual y la va rellenando dinámicamente según se van produciendo distintos eventos de memoria durante la ejecución del proceso actual, concretamente, fallos de página y operaciones de invalidación de entradas de la TLB. Téngase en cuenta que en esta solución se producirán un número significativo de fallos de página cuyo único objetivo es la construcción incremental de la tabla de páginas que gestiona el hardware de gestión de memoria, con la consiguiente sobrecarga. El nombre de esta técnica proviene de que la tabla de páginas que gestiona el hipervisor se va actualizando de forma dinámica según se vayan produciendo dichos eventos y, por tanto, tiene un comportamiento con una cierta similitud al de una TLB. A grandes rasgos, el comportamiento sería el siguiente:

- El sistema operativo alojado modifica el valor del *RBTP* para que haga referencia a la tabla de páginas del proceso actual (podría ser la primera vez que ejecuta o no). Se produce una excepción al ser una instrucción privilegiada y toma control el hipervisor que guarda dicho valor como parte del estado de la máquina virtual y construye una tabla de páginas vacía, que es a la que hace referencia el *RBTP* real. Como se verá más adelante, con esta técnica, después de un cambio de contexto, se comienza siempre con una tabla vacía. Nótese que como parte del estado de una máquina virtual el hipervisor guarda tanto el *RBTP* que hace referencia a la tabla de páginas que gestionará el hardware, como el valor que pretendió escribir el sistema operativo alojado en el *RBTP* (llamémosle *soRBTP*), que apunta a la tabla de páginas del proceso actualmente en ejecución construida por el sistema operativo alojado.
- Durante la ejecución de un proceso se produce un fallo de página porque ésta no está residente:
  1. Toma control el hipervisor que consulta la tabla de páginas construida por el sistema operativo alojado (puede acceder a la misma a través del *soRBTP*) y consulta el estado de la entrada correspondiente detectando que no está residente y propagando, por tanto, el fallo de página al sistema operativo.
  2. La rutina de tratamiento del fallo de página del sistema operativo realiza el procesamiento habitual y se completa actualizando la entrada correspondiente de la tabla de páginas que gestiona de manera que haga referencia al marco asignado dentro de la memoria física asociada a esa máquina virtual. Al completarse la rutina de tratamiento, el proceso vuelve a ejecutar la misma instrucción que vuelve a provocar un fallo de página puesto que en la tabla que gestiona el hipervisor la entrada sigue siendo inválida.
  3. Toma nuevamente el control el hipervisor ante el nuevo fallo de página, pero esta vez detecta en la tabla de páginas gestionada por el sistema operativo que la página está residente. El hipervisor copia la entrada de la tabla de páginas del sistema operativo a su tabla de páginas cambiando el número de marco para que haga referencia a la memoria real y eliminando el permiso de escritura de la página (más adelante, se explicará el motivo de esta restricción). Al completarse el tratamiento, el proceso vuelve a ejecutar la misma instrucción sin producirse un fallo esta vez.
- Si, después de un cambio de contexto, el proceso accede a una página que está residente pero aparece como inválida en la tabla de páginas que gestiona directamente el hipervisor, se realizará el mismo tratamiento que en el punto 3 del caso previo.

- Si el sistema operativo modifica alguna entrada de la tabla de páginas que gestiona correspondiente al proceso en ejecución actualmente (por ejemplo, la marca como no válida porque la región a la que pertenece ha sido invalidada) tendrá que ejecutar la instrucción que invalida esa entrada en la TLB. Al ser una instrucción privilegiada, toma control el hipervisor que invalidará la misma entrada en su tabla de páginas.
- Con esta solución, cualquier cambio que se haga sobre las tablas de páginas de un proceso que no esté ejecutando, no es detectada por el hipervisor, por lo que no tiene sentido que el hipervisor guarde tablas de páginas de los procesos que no están actualmente en ejecución. Por ello, cada vez que hay un cambio de contexto, el hipervisor crea una nueva tabla de páginas vacía para el proceso que va a ejecutar y elimina cualquier información que tuviera del anterior (comportándose nuevamente como una TLB).
- Un último aspecto a tener en cuenta es que, dado que el hardware no trabaja con las tablas de páginas que maneja el sistema operativo por cada proceso, sino con las que gestiona el hipervisor por cada máquina virtual, es necesario que el hipervisor propague explícitamente los bits de referencia y modificado a las tablas del sistema operativo. Es por eso que, como se explicó previamente, en el tratamiento del fallo de página, el hipervisor deja la página como de sólo lectura, de manera que si se produce un fallo adicional, es señal de que se está escribiendo en esa página, y en el tratamiento del fallo, el hipervisor puede propagar el bit de modificado a la tabla de páginas del proceso que gestiona el sistema operativo.

Para comparar esta solución con las restantes, a continuación se especifica qué estructuras de datos para la gestión de memoria se requerirían para la misma:

- Cada sistema operativo alojado gestionará una tabla de páginas para cada proceso, pero estas tablas no se instalan directamente en el hardware de gestión de memoria (cuando el sistema operativo alojado intenta modificar el *RBTP*, toma control el hipervisor).
- El hipervisor gestionará una única tabla de páginas por cada máquina virtual, que será la que se instala en el hardware de gestión de memoria y que se irá rellenando dinámicamente para reflejar el comportamiento del proceso actualmente en ejecución, según se ha explicado previamente en este apartado.
- El hipervisor mantendrá por cada máquina virtual una tabla, no vinculada con el hardware de gestión de memoria, que mantendrá las correspondencias entre los marcos de esa máquina virtual y los marcos de la memoria real asignados a los mismos.

Recapitulando, se trata de una solución que requiere el uso de una cantidad de memoria relativamente baja (una tabla de páginas adicional por cada máquina virtual) pero que, al no guardar información de las tablas de páginas de un proceso entre cambios de contexto, produce bastantes fallos de página para ir reconstruyendo la tabla después de cada cambio de contexto.

Tablas de páginas en la sombra

Esta solución requiere un gasto de memoria más elevado, ya que se almacena una tabla de páginas adicional (en la sombra) por cada proceso de cada máquina virtual, pero, en principio, reduce significativamente el número de fallos de página adicionales requeridos para implementar esta técnica. La idea en la que se basa es mantener sincronizadas la tabla de páginas del proceso que mantiene un sistema operativo alojado y la copia en la sombra de esa tabla que gestiona el hipervisor (evidentemente, en la primera tabla las entradas harán referencia a los marcos asignados a la máquina virtual, mientras que en la segunda corresponderán a los marcos de la memoria física a los que están asociados los primeros). La dinámica de esta técnica sería, a grandes rasgos, la siguiente:

- Cuando se produce un cambio de contexto a un proceso que no había ejecutado hasta entonces y el hipervisor toma control porque el sistema operativo alojado ha escrito en el *RBTP*, éste crea una copia de la tabla y cambia los permisos de la tabla que gestiona el sistema operativo alojado para impedir que se pueda modificar. Además, en la estructura de datos donde almacena el estado de cada máquina virtual debe guardar las direcciones físicas virtuales de las tablas de páginas de cada proceso, de manera que cuando el sistema operativo alojado intente modificarlas y se produzca una excepción, el hipervisor pueda identificar cuál es la tabla en la sombra asociada a la tabla que se intentó modificar a partir de la dirección del fallo de página.

- Cuando se produce un fallo de página porque la página no está residente, el hipervisor le propaga el fallo al sistema operativo alojado que realiza el procesamiento habitual y actualiza la entrada correspondiente de la tabla de páginas que gestiona, de manera que haga referencia al marco asignado dentro de la memoria física asociada a esa máquina virtual. Al intentar escribir en la entrada, se produce una excepción por estar protegida, tomando control el hipervisor que actualiza la entrada de la tabla en la sombra, pero de manera que haga referencia al marco de la memoria real asociado al marco especificado por el sistema operativo, dejándola como de sólo lectura.
- Cualquier cambio en una entrada de la tabla de páginas de un proceso (como, por ejemplo, poner el bit de referencia a 0 o expulsar una página), estando o no el proceso en ejecución, provocaría una excepción que recibiría un tratamiento similar al explicado en el punto anterior, copiándose la entrada desde la tabla de páginas que gestiona el sistema operativo.
- En un cambio de contexto a un proceso que ya ha ejecutado previamente, cuando toma control el hipervisor, debe asignar al *RBTP* la dirección donde está almacenada la tabla de páginas en la sombra que corresponde a la tabla de páginas gestionada por el sistema operativo del proceso a ejecutar.
- Un último aspecto a tener en cuenta es que, como ocurría en la solución previa, es necesario que el hipervisor propague explícitamente los bits de referencia y modificado a las tablas del sistema operativo, usando para ello la misma técnica, es decir, dejar la página como de sólo lectura y si se produce un fallo adicional, el hipervisor propagará el bit de modificado a la tabla de páginas del proceso que gestiona el sistema operativo.

Para comparar esta solución con las restantes, a continuación se especifica qué estructuras de datos para la gestión de memoria se requerirían para la misma:

- Cada sistema operativo alojado gestionará una tabla de páginas para cada proceso, pero estas tablas no se instalan directamente en el hardware de gestión de memoria (recuerde que cuando el sistema operativo alojado intenta modificar el *RBTP*, toma control el hipervisor).
- El hipervisor gestionará una tabla de páginas por cada proceso de cada máquina virtual, que será la que se instala en el hardware de gestión de memoria y que se mantendrá sincronizada con la tabla de páginas del mismo proceso que gestiona el sistema operativo alojado, según se ha explicado previamente en este apartado.
- El hipervisor mantendrá por cada máquina virtual una tabla, no vinculada con el hardware de gestión de memoria, que mantendrá las correspondencias entre los marcos de esa máquina virtual y los marcos de la memoria real asignados a los mismos.

Uso de procesadores con soporte para la virtualización

Dadas las dificultades a la hora de implementar una solución eficiente para la virtualización de la memoria sobre un procesador convencional debido a la necesidad de generar un número muy significativo de eventos adicionales (fallos de página, excepciones al escribir sobre una zona protegida,…) que tiene que tratar el hipervisor, algunos procesadores han incorporado técnicas que facilitan esta labor como son las tablas de páginas anidadas de AMD-V o extendidas en el caso de Intel VT-x, que son funcionalmente equivalentes.
La propuesta consiste en que el hardware de gestión de memoria, además de implementar la clásica traducción de direcciones lógicas a físicas, añade un segundo nivel que realiza la traducción de esas direcciones físicas, que serían las que maneja la máquina virtual, a las direcciones físicas reales, como se refleja en la siguiente figura extraída de http://developer.amd.com/wordpress/media/2012/10/NPT-WP-1%201-final-TM.pdf.

De esta manera, el hardware soporta directamente los dos niveles de traducción y no será necesario forzar eventos artificiales que le permitan al hipervisor tomar control para implementar la virtualización. El procesador dispondrá de un registro que hace referencia a las tablas de páginas de alto nivel (llamémosle *RBTPA*), encargadas de las traducciones de direcciones lógicas a físicas de la máquina virtual, y un registro que apunte a las tablas de páginas de bajo nivel (denominémosle *RBTPB*), a cargo de las traducciones de direcciones físicas de la máquina virtual a físicas reales. La dinámica de esta solución sería la siguiente:

- El sistema operativo alojado mantendrá de forma convencional una tabla de páginas por cada proceso, que será la que instalará en el hardware de gestión de memoria como tabla de alto nivel cada vez que a ese proceso le toque ejecutar. Tanto los cambios de proceso como el tratamiento de

los fallos de página y de las modificaciones de las entradas de las tablas de páginas las realizará el sistema operativo alojado de forma convencional, igual que en un sistema no virtualizado.

- El hipervisor mantendrá por cada máquina virtual una tabla, que se instalará en el hardware de gestión de memoria como tabla de bajo nivel cada vez que esa máquina virtual esté activa. Cuando el hipervisor tenga que reasignar el procesador a otra máquina virtual, tendrá que modificar, por un lado, el *RBTPB* para que haga referencia a las tablas de páginas de bajo nivel de la nueva máquina virtual y, por otro lado, el *RBTPA* de manera que apunte a las tablas de páginas del proceso en ejecución en dicha máquina.

Para comparar esta solución con las restantes, a continuación se especifica qué estructuras de datos para la gestión de memoria se requerirían para la misma:

- Cada sistema operativo alojado gestionará una tabla de páginas para cada proceso.
- El hipervisor mantendrá una tabla de páginas por cada máquina virtual.

En principio, parecería que esta solución es óptima en cualquier escenario. Sin embargo, hay que tener en cuenta que, aunque se ha eliminado la necesidad de generar eventos artificiales para realizar la virtualización, debido al uso de dos niveles de traducción, el coste de un fallo de TLB, que lo gestiona la propia MMU del procesador, es muy significativo, ya que requiere ni más ni menos que 24 accesos a memoria en un sistema que use tablas de páginas jerárquicas con 4 niveles, puesto que toda dirección física de una máquina virtual también tiene que ser traducida. Se deja como ejercicio al lector que determine cuáles son esos 24 accesos, sugiriéndole que revise la sección 4.2.2 de la URL recomendada previamente.