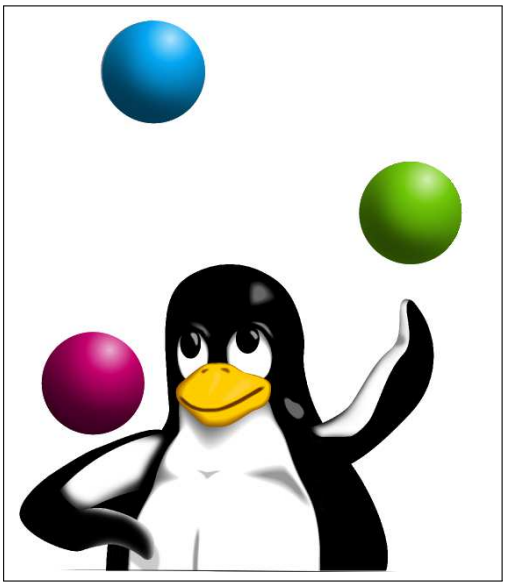
Virtualización



Esteban Ruiz
eruiz0@fceia.unr.edu.ar
(Basado en un apunte de Leonardo Scandolo)
Sistemas Operativos II - 2016
Departamento de Ciencias de la Computación
FCEIA-UNR

Índice

1. Conceptos básicos de virtualización

1.1. ¿Por qué existen máquinas virtuales?

El uso más frecuente es para aislar programas que brindan servicios. Muchas veces, dichos programas necesitan ambientes distintos (distintas versiones de librerías, S.O., etc) por lo cual es más fácil tener una computadora por programa.

Otro problema es que si ubicamos más de un servicio por computadora, un desperfecto en uno puede afectar el funcionamiento de otros servicios.

La virtualización resuelve todos estos problemas: cada servicio puede ejecutarse en su propia *máquina virtual*, con el entorno que necesite y aislado de manera que si se cae un servicio, los otros no son afectados.

1.2. Tipos de virtualización

Existen 2 tipos distinguibles de virtualización:

- Hipervisores (a veces llamados hipervisores tipo 1): son una capa fina de software que se activa sólo cuando el S.O. virtualizado quiere acceder al hardware. Ejemplos: VMWare Workstation, Linux KVM, Microsoft Hyper-V.
- Emuladores (a veces llamados hipervisores tipo 2, o monitores de máquinas virtuales): son programas de usuario que emulan las instrucciones que acceden al hardware. Ejemplos: VMWare Server, Microsoft Virtual Server, Parallels Desktop, VirtualBox.

1.3. Conceptos fundamentales

El S.O. virtualizado es llamado huésped (guest) y, en el caso de los emuladores, el S.O. donde se ejecuta el emulador es llamado anfitrión (host).

Un concepto fundamental es el de *instrucciones sensibles*. Estas instrucciones son todas las que acceden directamente al hardware de la computadora (fuera de la memoria y el ALU). Generalmente sólo puede ejecutarlas el S.O. pues requieren privilegios especiales (p. ej.: modo supervisor).

Para poder virtualizar una computadora es necesario que el S.O. huésped se ejecute enteramente en modo usuario: en caso contrario apareceran todo tipo de conflictos (por ejemplo: al intentar utilizar puertos, discos rígidos, interrupciones, etc...). Cuando un S.O. quiera ejecutar una operación sensible, se debería saltar a modo kernel. La arquitectura de las computadoras 386 y sus derivadas no hacían esto hasta el 2005: algunas instrucciones sensibles simplemente se ignoraban si se ejecutaban en modo usuario. Por lo tanto, no podían usarse hipervisores para estas arquitecturas, sólo emuladores.

A partir del 2005, Intel con VT (Virtualization Technology) o VT-x para x86 y AMD con SMV (Secure virtual machine) y luego AMD-V, agregan funcionalidad que permite la virtualización con el uso de hipervisores.

2. Hipervisores

Los hipervisores son básicamente sistemas operativos muy pequeños que ejecutan a los S.O. huéspedes como procesos de usuario.

Cuando un S.O. huésped quiere ejecutar una operación sensible, el hilo de ejecución pasa al hipervisor, que ejecuta la operación para el S.O. huésped, haciendo las traducciones que deba hacer.

3. Emuladores (o hipervisores tipo 2)

Los emuladores como VirtualBox o VMWare Server no pueden confiar en que una instrucción sensible genere un salto a modo kernel, por lo cual deben emular el funcionamiento y los efectos de estas instrucciones.

Lo que se hace es la traducción de las partes de código que son sensibles, lo que se llama BT (binary translation, o traducción de binarios).

La manera de hacerlo tradicionalmente es ir dividiendo el código a ejecutar en *bloques*. Cada bloque es un conjunto de instrucciones que no son ni saltos ni instrucciones sensibles, y que terminan en alguna de esas instrucciones. Cada bloque se *cachea* y se ejecutan todas las instrucciones normalmente menos la última en modo usuario. La última instrucción se emula porque puede necesitar la intervención del emulador. Una vez que se emula la última instrucción, se sigue con la ejecución normal del próximo bloque.

Uno pensaría que este tipo de virtualización es más lenta que la virtualización hecha por hipervisores, dado que hay que crear los bloques en tiempo de ejecución y emular las instrucciones en modo usuario. Sin embargo, este esquema de virtualización a veces es más rápido debido al cacheo de instrucciones y a que no se salta a modo kernel tan seguido.

4. Paravirtualización

Hasta ahora vimos maneras de virtualizar un S.O. que ignora que está ejecutándose sobre una máquina virtual. Sin embargo, debido a que el código fuente de muchos S.O. están disponibles para su uso y modificación, es posible modificarlos para que se ejecuten sobre un hipervisor específico. La idea es que el hipervisor provea una API para acceder al hardware simulado, y que el S.O. huésped sea modificado para hacer uso de esta API en vez de llamar a operaciones de I/O. De esta manera se hace más sencillo (y menos propenso a bugs) escribir hipervisores, y se hace más rápida la ejecución del S.O. huésped.

Un hipervisor de paravirtualización es prácticamente un microkernel que ejecuta S.O. huéspedes como procesos.

Para estandarizar la API de paravirtualización se creó VMI (Virtual Machine Interface), que es la especificación de una API standard para virtualización.

Un ejemplo del uso de VMI es VMIL (VMI Linux), que es un kernel modificado para hacer llamadas VMI en vez de operaciones sensibles y un ejemplo de hipervisor de paravirtualización es Xen.

5. Virtualización más allá de las instrucciones

5.1. Virtualización de memoria

El problema más grande que se plantea en virtualización de memoria es cómo mapear las páginas físicas que el S.O. huésped cree que existen con páginas reales, o sea que debe

haber un segundo nivel de traducción: página virtual huésped → página física huésped → página física anfitrión.

El hipervisor lleva una *shadow* page table con la traducción de las páginas que le dio al S.O. huésped. Esto es fácil de hacer y es una de las características que proveen las tecnologías de virtualización de Intel y AMD.

¿Como se hace con la TLB? Una forma es flushear la TLB cada vez que se pasa de una VM a otra, pero esto hace que se tenga menor desempeño, dado que hay que llenar la TLB de nuevo a cada cambio de contexto.

¿Como resuelve AMD-V el problema de eficiencia de usar shadow page tables y flushear la TLB a cada cambio de contexto?

Para la traducción de páginas se desarrolló una extensión en hardware llamada NPT (Nested page table), que es básicamente la shadow page table en hardware, junto con RVI (Rapid Virtualization Indexing), que traduce de página virtual a página física real directamente en hardware y por lo tanto mucho más rápido que usando shadow page tables en software.

En el sistema AMD-V, se agrega un parámetro a la TLB, el ASID (adress space identifier), esta TLB se llama tagged TLB. A cada VM se le asigna un ASID único que conoce sólo el hipervisor, pero que el S.O. huésped no puede acceder. Al volver de un cambio de contexto, si no se usaron todas las entradas de la TLB, las que quedaron se pueden usar nuevamente.

¿Como resuelve Intel VT-x estos mismos problemas? Los procesadores (VT-x) de Intel poseen lo que es llamado EPT (Extended Page Table), que proveen la misma funcionalidad que las NPT de AMD, y además ayudan a virtualizar el uso de DMA (direct memory access), y espacios de memoria de I/O mapeados a memoria.

5.2. Virtualización de I/O

Las funciones de I/O son siempre sensibles, y por eso siempre son manejadas por el hipervisor o emulador.

El S.O. huésped tratará de escribir en las posiciones de memoria donde cree que un dispositivo puede leer (a través de DMA). Sin embargo, estas direcciones no son reales, por lo cual el hipervisor debe traducir estas llamadas para escribir realmente en la posición física donde el hardware pueda leerlo. De la misma manera cuando un dispositivo escribe en una posición determinada por DMA, se debe copiar, o mapear esa dirección física a una dirección que pueda leer el S.O. huésped. Este proceso es lento y proclive a introducir errores.

Para mejorar esto, ambos AMD e Intel definieron en sus tecnologías de virtualización recursos para poder realizar estas tareas asistidas por hardware. Estas tecnologías extienden las capacidades del componente que maneja la memoria mapeada a dispositivos, que se llama normalmente IOMMU (I/O Memory Management Unit)

En AMD, se le llama AMD-Vi, y permite dividir la memoria asignada a un dispositivo I/O en varias partes. Cada parte se asigna a una máquina virtual particular, y en hardware se hace la traducción entre las direcciones del huésped y las físicas, además de asegurarse de que una VM no escriba en la memoria asignada a otra.

En Intel, se le llama VT-d, y también subdivide la memoria de I/O en dominios de protección, que cumplen la misma función que la tecnología AMD-Vi.

Algunos hipervisores basados en paravirtualización (como Xen) tienen una máquina virtual dedicada a ejecutar instrucciones I/O. De esta manera el hipervisor pasa a ser

una especie de microkernel y dicha máquina virtual dedicada es el sistema de archivos.

Referencias

[1] Chris Wolf, *Virtualization: From the Desktop to the Enterprise*

[2] Clark Scheffy, *Virtualization for dummies*

[3] *VMWare virtualization online resources, en http://www.vmware.com/virtualization*