Virtualización

Gunnar Wolf

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

¿Qué significa *virtualizar*?

- Proveer algo que no está allí, aunque parece estarlo
- Ofrecer y mantener una ilusión
  - Un truco de magia

La *virtualización* es, en términos generales, ofrecer recursos que no existen en realidad — Y mantener la ilusión, tan bien como sea posible.

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Ámbitos de virtualización

- Es un término *de moda*, que nos encontraremos cubriendo muy distintas tecnologías
- Lleva existiendo –de diferentes maneras– muchas décadas
- Cubriremos algunas estrategias y tecnologías de virtualización comunes hoy en día
  - Con diferentes usos y propósitos
  - Muchos de los cuales utilizamos día a día sin pensar en ello

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

¿Diferentes tecnologías?

- Muchas cosas pueden ser entendidas por *virtualización*
- Hay muchos diferentes casos de uso, y cada uno requiere una solución diferente
- Incluso para un mismo caso de uso, hay más de una manera de llegar al mismo resultado
  - Hay espacio para que la *selección natural* haga su trabajo
- Las diferentes tecnologías no tienen líneas divisorias tan claras
  - Un proyecto pueden caer en varias clasificaciones
  - O ser originalmente de un tipo, e ir migrando *naturalmente* hacia otro

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

¿Qué es *emular*?

- La técnica de virtualización disponible hace más tiempo en *computadoras personales*
- El *procesador anfitrión* traduce cada una de las instrucciones, simulando *en tiempo de ejecución* hardware inexistente
- Fue muy popular hacia la segunda mitad de los 1980 y a principios de los 1990, durante la explosión de las arquitecturas
- Es *altamente* ineficiente — Resulta muy caro en tiempo de cómputo

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

¿Emular o simular?

- Un *emulador* busca *imitar* el *comportamiento completo* del sistema destino
  - Emular: *Imitar las acciones de otro, procurando igualarlo o superarlo* (WordReference.com)
  - Reproduce *todos los procesos internos*
  - Implementa *los mismos* mecanismos
- Un *simulador* simula o finge las *áreas de interés* del sistema destino
  - Puede emplear datos para generar respuestas predefinidas, obviando (brincándose) procesos

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

¿Emular o simular? Un ejemplo

Un *simulador de vuelo* no nos lleva a ningún lugar, aunque presente una cabina como la de un avión real

Un *emulador* busca ejecutar software arbitrario *sin que éste detecte la diferencia*

Emulación de una arquitectura existente

- Se puede hacer a diferentes profundidades
  - Desde emular el sistema completo (juego de instrucciones, chipset, buses, etc.)
  - Hasta emular únicamente *parte* del chipset (muy común en arquitecturas m680x0)
- La arquitectura *Amiga* de Commodore es la primera de uso personal en ofrecer varios programas emuladores
  - Macintosh y Atari ST (misma plataforma m680x0) a velocidad nativa
  - Plataforma PC, pero *muy, muy* lenta (incluso XT 8088)

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Utilidad actual de la emulación

- A difrencia de lo que ocurría hace 20 años, hoy en día este tipo de emulación es muy socorrido en el "mundo real"
- Los sistemas *embebidos* son cada vez más comunes
  - Computadoras pequeñas, limitadas en recursos (memoria, almacenamiento, velocidad)
  - Diseñadas para correr con el menor consumo energético posible
  - Aún a costa de un menor rendimiento
- Celuluares, cámaras, ruteadores, scanners, controladores de equipo industrial...
  - Parte muy importante del mercado
- Emular m680x0 o ARM en un buen equipo de escritorio resulta en velocidad comparable al hardware nativo

Emulando arquitecturas inexistentes

- También podemos emular una arquitectura que *nunca ha sido implementada*
- La idea viene también de los 1970
  - En pos de la portabilidad, UCSD definió un *p-system*, a ser ejecutado en una *p-machine*
  - Esta computadora nunca existiría en realidad, pero varias arquitecturas existentes ofrecerían *emuladores de p-machines*
- La arquitectura de la *p-machine* está definida en torno al lenguaje Pascal

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Emulando arquitecturas inexistentes

- Todo programa hecho para correr en una *p-machine* correría en cualquier arquitectura que lo implementara
- Los *p-systems* gozaron de relativa popularidad hasta mediados de los 1980, con implementaciones en arquitecturas 6502, Z80 y 80x86

Arquitecturas emuladas, de uso diario — E inexistentes

- En la década de los 1990, Sun Microsystems retomó las ideas de los *p-systems*, y diseñó la arquitectura *Java*
- Java está pensado para ser una arquitectura idealizada
  - Nativamente orientada a objetos
  - Buscando dar una completa *portabilidad* al código
  - Slogan: *Write Once, Run Anywhere*
- Microsoft retomó varios años más tarde esta misma idea, creando la arquitectura *.NET*
  - Su principal contribución es plantear a la máquina virtual como independiente del lenguaje de programación
- Desde el 2000, las comunidades (principalmente) de *Perl* y *Python* han implementado *Parrot*
  - Máquina virtual apta para lenguajes de script

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Esquema de la arquitectura *.NET*

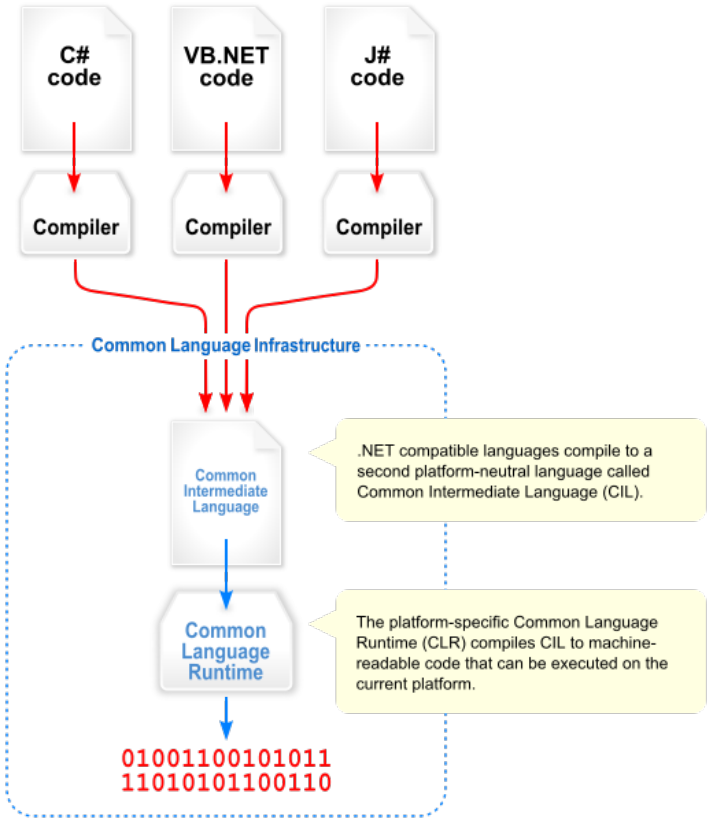

Figura: Arquitectura de la infraestructura de lenguajes comunes (CLI) de .NET (Imagen: Wikipedia)

¿Por qué utilizar/emular estas arquitecturas?

- Las abstracciones presentadas por estas máquinas virtuales resultan demasiado complejas para ser implementadas directamente en hardware
- Son, sin embargo, muy útiles al programador, que sabrá sacarles buen jugo
- Sun diseñó la arquitectura *MAJC* (1999) para ejecutar directamente código Java
  - Los chips resultaban demasiado complejos y caros
  - Fracaso comercial
- MAJC implementaba una arquitectura VLIW y optimización basada en múltiples hilos de ejecución
  - Ideas retomadas para generaciones actuales de CPUs

Transmeta: El procesador emulador

- En el 2000, *Transmeta* anunció su procesador *Crusoe*, orientado al mercado de bajo consumo energético
- Su arquitectura está diseñada para ejecutar código *diseñado para otras arquitecturas*
  - Traducido a través del microcódigo: *Code Morphing Software* (*Software de transformación de código*)

Transmeta: El procesador emulador

- La única arquitectura implementada en CMS es la Intel x86
  - Pero las dos generaciones de procesadores Transmeta (Crusoe y Efficeon) son completamente distintas
  - Gracias a CMS, esta difrencia es transparente al usuario
- Tecnología muy interesante, y aplicada ya fuera de Transmeta
  - A pesar de esto, Transmeta colapsó como empresa.

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

La emulación, mejorada

- Las técnicas utilizadas para la emulación han mejorado tremendamente en los úlitmos diez años
- Los emuladores hacen hoy traducción predictiva y compilación del código a ejectuar a formatos nativos (traducción dinámica)
- También guardan copias convertidas/compiladas del código a emular
  - *Compilador JIT — Just in Time*; *Compilador Justo a Tiempo*

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

La emulación, mejorada

- En líneas generales, la vieja fama de la lentitud de las máquinas virtuales ya no se justifica
- Las máquinas virtuales pueden llamar a código nativo para puntos críticos donde haga falta optimización
- . . . Y las usamos transparentemente, todos los días

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Virtualización asistida por hardware (HVM)

- Buena parte del *ruido* que hoy en día recibe la virtualización es a consecuencia de esta modalidad
- Algunas arquitecturas de cómputo incluyen provisiones para ser *virtualizadas*
  - Especialmente máquinas diseñadas como *grandes*
- Primer ejemplo: IBM S/360-67
  - Sistema operativo CP-67/CMS (1968-1972)
  - Sistema operativo ligero, monousuario
  - Pensando en que siempre habría *múltiples instancias* del sistema operativo en ejecución bajo el *hipervisor* CP
- Computadoras inherentemente de *tiempo compartido*, dando a sus usuarios la *ilusión* de tener una computadora dedicada a ellos

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

La motivación detrás de CP67/CMS

- Motivación de la virtualización:
  - Maximizar el aprovechamiento de recursos
  - Proveer administración centralizaa
- Al virtualizar el sistema *completo*, este sistema ofrece mayor aislamiento, seguridad y confiabilidad que cualquier sistema de *tiempo compartido*
- Permite además correr cualquier programa diseñado para una máquina S/360, incluso si no estaba diseñado para tiempo compartido
- IBM reimplementó este sistema como VM/370, al contar con una arquitectura de memoria virtual
- z/VM, derivado de este, sigue ampliamente en uso hoy en día

CP/CMS y VM como software libre

- CP/CMS fue distribuido directamente como código fuente
- Desde un principio se desarrolló una activa comunidad de usuarios estudiando y modificando el código fuente
- Por fricciones políticas dentro de IBM, tanto VM com CP/CMS fueron también distribuídos como parte de las *bibliotecas no soportadas* en la colección *Type-III*
- Hoy en día se pueden bajar estos sistemas y ejectutarlos
  - Dentro del emulador *Hercules* de sistemas S/370, S/390 y zSeries

El *hipervisor*: Más abajo que el núcleo

- Tradicionalmente las arquitecturas virtualizables corren un micro-sistema operativo encargado de gestionar a *cada uno de los sistemas operativos* que corre en cada una de las máquinas virtuales
  - Es un *micro* SO porque no cubre muchas de las áreas clásicas (sistemas de archivos, comunicación entre procesos, gestión de memoria virtual, . . .)
  - Se limita a gestión básica de memoria física contigua, asignación de dispositivos, y *poco* mas que eso
  - Ojo: Hay hipervisores que son sistemas operativos completos (como KVM bajo Linux)
- Este micro-SO es conocido como el *hipervisor*
  - Dando a entender que hace más que *supervisar*, el rol tradicional del SO

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Hipervisor oculto

- Idealmente, el núcleo de cada una de las máquinas virtuales *no sabe siquiera* que está siendo ejecutado dentro de un hipervisor
  - La ilusión es completa
- En algunas arquitecturas puede incluso haber múltiples niveles de hipervisores

El panorama hasta ≈ 2005

- Las arquitecturas que proveían virtualización por hardware eran muy especializadas
  - Muy caras, fuera del alcance de los usuarios en general
  - Fuera del alcance incluso de la mayor parte de los desarrolladores
- En 2005, Intel lanza la *Vanderpool Technology* para sus procesadores x86 (extensión *VT-x*)
- En 2006, AMD lanza los procesadores con *extensiones Pacifica*
- Hoy en día, casi todas las computadoras de escritorio rango medio-superior vienen con soporte para HVM
- El tema era tan novedoso que tardó algunos años en desarrollar tracción

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Estabilidad por virtualización

- Es aceptado universalmente que la mayor parte fuente de inestabilidad en los sistemas operativos son los drivers
  - Es código típicamente más *sucio* que el de otras partes del núcleo
  - Proviene de todo tipo de fuentes, desde desarrolladores independientes hasta las compañías desarrolladoras del hardware
- Dando control de calidad a los manejadores de los dispositivos emulados/virtualizados, podemos lograr que los sistemas operativos huésped sean *más estables* de lo que serían sobre el hardware real

Estabilidad por virtualización

- Típicamente el hipervisor ofrecerá a los huéspedes dispositivos relativamente viejos y simples
  - Red NE2K
  - sonido Soundblaster16
  - video Cirrus
  - ...
- Las prestaciones máximas del hardware *no están limitadas* por las características del hardware emulado — Sólo su interfaz al sistema

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

HVM, pero sin hardware

- La primer versión públicamente descargable de VMWare fue liberada en 1998
  - Por parte del software libre, `kqemu` fue presentado (gratuito pero no-libre) en 2005, y liberado bajo la GPL en 2007
- Agrega a la arquitectura x86 clásica las funciones básicas de HVM, sin implementarlas por hardware
  - Con una notable penalización en velocidad
  - Aunque *muchísimo* menor a la de la emulación
- Como la mayor parte del código sigue siendo nativo, ofrecen una velocidad general muy aceptable

¿Cómo puedo implementar HVM?

- Proyectos libres:
  - Xen (modo *asistido por hardware*)
  - KVM (sobre Linux)
  - Logical Domains (sobre Solaris)
- Proyectos híbridos libre/propietario
  - VirtualBox
- Productos propietarios
  - VMWare
  - VirtualPC
  - HyperV
  - Parallels
- . . . Y seguramente muchos más

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Un enfoque más ligero, más accesible

- Aún si la virtualización asistida ya está disponible en CPUs disponibles masivamente, es aún una característica *de lujo*
  - Para los rangos superiores del mercado
- La *paravirtualización* consiste en reescribir las porciones de un sistema operativo que interactúan directamente con el hardware, para que soliciten estas operaciones *a sabiendas* a un hipervisor
- Es conocida como *virtualización asistida por el sistema operativo* (*OS-assisted virtualization*)
- Formalmente podría verse como un *port* del sistema operativo a una nueva arquitectura
  - *Muy* parecida a la del sistema anfitrión

Paravirtualización y software libre

- Si bien ofrece un mapeo más directo, mejor rendimiento y más estabilidad a los sistemas huésped, requiere modificaciones bastante amplias al sistema operativo
  - Es prácticamente imposible correr sistemas no-libres paravirtualizados
- Un sistema operativo tiene que ser *portado* a las abstracciones que ofrece cada una de las arquitecturas de paravirtualización
- El artículo con el cual se presentó Xen 1.x habla de un port de Windows XP, basado en el Academic Licensing Program a su paravirtualizador
  - Pero no es redistribuible, sólo puede ser utilizado internamente en Xensource

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Aprovechamiento de recursos (1)

- Con sistemas paravirtualizados podemos lograr un consumo de recursos aún más eficiente que en un sistema virtualizado "real"
- Los dispositivos presentados al OS huésped son mucho más ligeros e *idealizados*
  - No hace falta emular al hardware real

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Aprovechamiento de recursos (2)

- El OS huésped puede pedir al anfitrión recursos adicionales cuando los requiere
  - Incluso sobre demanda — *balooning*
  - Incluso recursos que para una computadora normal son inamovibles, p.ej. espacio de memoria
- Puede haber un monitoreo mucho más completo
  - El OS anfitrión no tiene que adivinar tantos detalles del funcionamiento del huésped si éste se los confía

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

CPU virtual, dispositivos paravirtuales

- Este punto es empleado por todo tipo de virtualizadores: Paravirtualización a nivel dispositivo
- Entre más sencillos sean los dispositivos emulados para la virtualización, menos sobrecarga por traducir llamadas a hardware inexistente
- Hasta una interfaz tan simple como NE2K tiene hardware innecesario a la hora de virtualizar
  - Entre más delgada sea la capa de traducción, mejor rendimiento obtenemos
- En Linux, las clases de dispositivos `virtio` y `pv` llegan a ofrecer rendimiento de 5 a 10 veces mejor que la emulación de dispositivos reales

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Sugerencia bibliográfica

Los temas presentados en este apartado están descritos muy bien y muy a detalle en el artículo *Xen and the Art of Virtualization* (`http://www.cl.cam.ac.uk/netos/papers/2003-xensosp.pdf`), Paul Barham, Boris Dragovic et. al. 2003
La progresión es muy natural y sencilla, ¡vale la pena al menos echarle un ojo!

¿Cómo puedo implementar paravirtualización pura?

- La principal arquitectura para esto es Xen
- VMWare ofrece un modo de operación basado en la paravirtualización

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Xen y KVM: Los dos competidores libres

- Las dos principales implementaciones libres de virtualización son Xen y KVM
- Sus ofrecimientos son en buena medida comparables
- Hasta $\approx$ 2010 parecía que KVM terminaría conquistando el terreno en que el anfitrión/hipervisor es Linux
  - Admitido mucho antes al *kernel oficial*
  - Xen requirió cambios mucho más profundos
- Hoy en día, ambos ofrecen interfaces bastante completas

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Xen: Un hipervisor mínimo

- Xen es un hipervisor puro — GRUB llama al núcleo de Xen, y éste lanza a un núcleo Linux
  - Este núcleo tiene que estar compilado para correr *paravirtualizado* a la arquitectura virtual de Xen
  - Esta primer máquina virtual tendrá *control del hipervisor*
  - En lenguaje de Xen, es `Dom0`
  - Se comunica con Xen a través del demonio `xend`
- Todas las máquinas virtuales adicionales que lancemos son `DomU`

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

KVM: El Linux de siempre... Mas un módulo raro

- KVM agrega funciones de hipervisor al núcleo estándar de Linux
  - A fin de cuentas, un sistema operativo completo tiene todo lo necesario para gestionar recursos entre diferentes procesos en ejecución
- Hereda / incluye *muy* buena parte de Qemu
- Las diferentes máquinas virtuales son sencillamente más procesos dentro del árbol de procesos

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

. . . ¡Cuéntenme ustedes al respecto!

¡Viva! ¡Hurra!
¡Ya era hora! ¡Tenemos tarea!

- Lean *Notes from a container* (Jonathan Corbet, 2007; `http://lwn.net/Articles/256389/`)
- Hagan un *mapa mental* (entregar dibujado/impreso, en papelito tradicional)
- Para el *martes 27 de agosto*
- *Sugerencias*: Leer e incluir conceptos tocados por los comentarios y en las ligas

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

¿Qué es un contenedor?

- Una manera distinta de virtualizar
- Más sutil, menos flexible
- Empleando *un mismo núcleo* de sistema operativo
  - Empujando la virtualización *una capa* hacia arriba
- Con mayores limitantes, pero importantes ventajas

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Herederos de `chroot`

- Los sistemas Unix han ofrecido la llamada al sistema `chroot` desde 1982
- Bill Joy la introdujo cuando trabajaba en 4.2BSD para probar la construcción de nuevas versiones del sistema operativo sin modificar el sistema *vivo*
- `chroot` permite *encerrar* a un proceso dentro de un directorio
  - Un proceso al que se le aplica `chroot` no puede ver el sistema de archivos fuera del directorio especificado
  - . . . No sin aplicar algunos trucos
- `chroot` sólo afecta la *visión de la raiz del sistema de archivos*
  - No es (ni busca ser) un verdadero aislamiento

No lo es... ¿Por qué no lo adecuamos?

- Los *contenedores* construyen sobre `chroot`, ampliando el aislamiento a otros componentes del sistema
- El primer sistema en ofrecer esta facilidad fue FreeBSD, con sus *jails*, desde la versión 4.0 (2000)
- Están también implementados ahora en Linux (*vserver* desde 2002, hoy *lxc*), Solaris 10 en adelante (*Zones*, 2005) y NetBSD/FreeBSD (*Sysjail*, utilizando `systrace`, 2006)
- Idea similar en Windows: *Parallels Virtuozzo*

Principios básicos de los contenedores

- La nomenclatura básica cambia según la implementación
  - Cada *servidor virtual* puede llamarse *contenedor*, *contexto de seguridad*, etc.
- El kernel oculta y aísla la información de cada contexto de los demás:
  - Tablas de procesos
  - Señales, IPC
  - Conexiones, sockets e interfaces de red, reglas de firewall
  - Dispositivos
  - Límites en consumo de recursos (RAM, CPU)
- Formalmente, los contenedores no implementan *virtualización*, sino *restricción*
  - Pero brindan al usuario la *ilusión* de una *máquina virtual* inexistente

Variedad, pero con un límite

- A través de los contenedores, la virtualización es *casi* completa
- Se ven un poco las *costuras*, pero para propósitos prácticos, cada *contenedor* es un sistema independiente
  - Excepto por el núcleo
- Podemos tener cualquier distribución corriendo dentro de nuestros contenedores al mismo tiempo
- Única restricción: Todos corren con el mismo núcleo (misma versión, mismos módulos, etc.)

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Consumo de recursos óptimo

- Un contexto sin actividad tiene un consumo de recursos mínimo
- Los procesos que no tienen actividad no consumen CPU
- Los procesos en memoria inactivos van siendo paginados a disco
- Queda como excepción *Sysjail* (OpenBSD), una implementación de contenedores en espacio de usuario a través de `systrace`, que *sí* es notablemente más lenta que el sistema en hardware nativo

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Algunos casos comunes de uso

- Mejor aprovechamiento / consolidación de recursos
- Migraciones
- Seguridad
- Redundancia / alta disponibilidad
- Despliegue de escritorios virtuales
- Simplificación de mantenimiento
- Desarrollo (especialmente depuración) para sistemas embebidos

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Diferentes necesidades, diferentes soluciones

- Hay una gran riqueza en la oferta de herramientas de virtualización
- Cada herramienta y estrategia tiene características muy distintas
- Muchas de las ofertas de cómputo *en la nube* cruzan necesariamente por virtualización
  - Particularmente por hardware, paravirtualización y contenedores

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

¿Y la nube?

- La *infraestructura como un servicio*, una de las modalidades del *cómputo en la nube* implica necesariamente virtualización
- Varios programas de administración de *nubes privadas* gestionan y monitorean también sistemas virtualizados
- Una *nube* puede verse como un conjunto de servidores configurados para brindar recursos reales a máquinas virtuales
  - Disco, memoria, tiempo de procesamiento, etc.
- La administración de servicios *en la nube*, así como sus ventajas y desventajas, salen del ámbito del curso
  - Pero sin duda será de interés de varios de ustedes

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

En esta presentación vimos. . .

- Emulación
  - Traducción de las instrucciones de una arquitectura diferente
  - Incluye la emulación de un sistema entero (hardware inexistente)
  - *Muy* lenta
  - Diferencia entre *emular* y *simular*
  - Puede emularse una arquitectura existente o inexistente
    - Modelo general del *p-system*, la *Máquina Virtual Java* (JVM), *Common Language Infrastructure* (.NET)
  - Rendimiento — Traducción dinámica, compilación JIT

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

En esta presentación vimos. . .

- Virtualización asistida por hardware (HVM)
  - Historia: IBM S/360-67 con CP-67/CMS (hasta hoy con z/VM)
  - Concepto de *hipervisor*
  - Entrada de HVM al común de los equipos x86
  - Estabilidad por virtualización

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

En esta presentación vimos. . .

- Paravirtualización
  - Enfoque más ligero, aunque requiere reescribir porciones de los sistemas operativos huésped
  - Un SO paravirtualizado puede lograr mejor uso de recursos que el mismo OS en hardware real
    - El hardware *idealizado* es más fácil de manejar
    - El sistema anfitrión no tiene que desperdiciar proveyendo recursos no empleados por los huéspedes
  - Hardware virtualizado, dispositivos paravirtualizados

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

En esta presentación vimos. . .

- Contenedores
  - Manera más *sutil*, menos flexible de virtualización
  - Un mismo sistema operativo (un mismo núcleo); un sistema maestro, varios *contenedores* con sistemas completos dentro
  - Construyendo sobre `chroot`
  - El núcleo separa varias estructuras presentando vistas separadas a los distintos sistemas huésped
  - Formalmente, más que virtualización implementan *restricción*
  - Consumo de recursos mínimo

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM