Estrategias de virtualización con Linux

Gunnar Wolf — gwolf@gwolf.org
http://www.gwolf.org/soft/virt

Instituto de Investigaciones Económicas, UNAM
Desarrollador del proyecto Debian

Encuentro Nacional de Linux y Software Libre
Octubre 25, 2008

Temas

¿Qué significa *virtualizar* en el cómputo?

- Proveer de 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.

¿Qué entendemos por virtualización?

- La virtualización es uno de los términos de moda hoy en día — Pero lleva existiendo de diferentes maneras por muchas décadas
- En esta presentación 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.

¿Diferentes tecnologías?

- Como vimos en un principio, muchas cosas pueden ser entendidas por virtualización
- Hay muchos diferentes casos de uso, y cada uno requiere una solución diferente, adecuada
- Incluso para un mismo caso de uso — Hay más de una manera de llegar al mismo resultado. Y debemos permitir que la *selección natural* haga su trabajo.
- Las diferentes tecnologías no tienen líneas divisorias tan claras; un proyecto puede caer en varias clasificaciones, o caer en una e ir migrando hacia otra

Temas

Emulación

- 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 en la segunda mitad de los 1980 y a principios de los 1990, durante la explosión de las arquitecturas
- Es, sin embargo, *altamente* ineficiente — Resulta muy caro en tiempo de cómputo.

Emulación de una arquitectura existente

- Se puede hacer a diferentes profundidades — Desde emular el sistema completo (desde el juego de instrucciones) 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 — Emular Macintosh y Atari ST (misma plataforma m680x0) funcionaba a velocidad nativa... Pero la emulación de PC (incluso emulando sólo el XT 8088) era ridículamente lenta

Utilidad actual de la emulación

- A diferencia 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), y diseñados para correr con el menor consumo energético posible, aún a costa de un menor rendimiento
- Celulares, PDAs, cámaras, scanners mecánicos, controladores de equipo industrial... Una rama muy importante del mercado actual
- Emular m680x0 o ARM en un procesador estándar de escritorio llega a ser más rápido incluso que el hardware nativo.

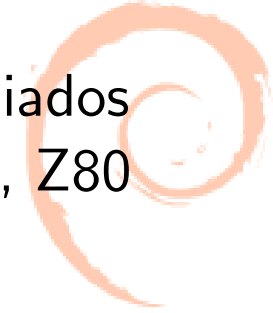
Emuladores libres

Algunos emuladores disponibles en Debian:

`qemu` (multiplataforma), `apple2`, `atari800`, `dosemu` (aplicaciones MS-DOS modo real), `e-uae` (Amiga m680x0, sin MMU), `nestra` (Nintento NES), `pose` (PDAs Palm), `specemu` (ZX Spectrum 48k), `stella` (Atari 2600), `vice` (Commodore PET, VIC20, 64, 128, CBM-II, PLUS/4), `xtrs` (TRS-80), `aranym` (Atari ST), `coldfire` (Frescale Coldfire 5206), `dosbox` (aplicaciones MS-DOS modo real con soporte de gráficos), `hatari` (Atari STe), `hercules`(IBM System/370, ESA/390, z/Architecture), `pearpc` (PowerPC), `simh` (Equipos históricos DEC, Honeywell, HP, IBM y otros), `dgen` (Sega Genesis/MegaDrive)

Emulando arquitecturas inexistentes

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

- Hay arquitecturas que han sido concebidas exclusicamente para propósitos académicos
- Donald Knuth diseñó la arquitectura MIX en los 1960 como *arquitectura ideal* para los ejemplos y ejercicios de su célebre libro *The Art of Computer Programming*, y su sucesora MMIX en 1999
- Es una arquitectura apta para la enseñanza, pero inviable para un sistema real. MIX Plantea un sistema híbrido binario-decimal, de 6 bits en modo binario o 2 dígitos en modo decimal; MMIX es una arquitectura RISC con 256 registros de 64 bits
- Existen MIXWARE/MMIXWARE —a emuladores (incompletos) de MIX y MMIX.

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
- *Write Once, Run Anywhere*
- Microsoft retomó varios años más tarde esta misma idea, creando la arquitectura .NET — Su principal contribución es el estar planteada como independiente de lenguaje
- Desde el 2000, las comunidades (principalmente) de Perl y Python están implementando Parrot, una máquina virtual apta para lenguajes de scripts

¿Por qué utilizar/emular estas arquitecturas?

- Las abstracciones presentadas por estas máquinas virtuales resultan demasiado complejas para ser implementadas directamente en el hardware
- Son, sin embargo, muy útiles al programador, quien sabrá sacarles muy buen jugo
- Sun diseñó una arquitectura de chips, MAJC (1999), para ejecutar directamente código Java — Pero los chips resultaban demasiado complejos y, por tanto, caros
- MAJC implementaba una arquitectura VLIW y optimización basada en múltiples hilos de ejecución — Estas ideas han sido retomadas en las 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 —a través del *Code Morphing Software*— código diseñado para otras arquitecturas
- La única arquitectura implementada en CMS es la Intel x86, pero las dos generaciones de procesadores Transmeta (Crusoe y Efficeon) son completamente distintas, y gracias al CMS, esto es transparente al usuario
- ...Es tecnología muy interesante, y se ha aplicado ya fuera de Transmeta. Sin embargo, personalmente dudo de que Transmeta (como compañía independiente) tenga viabilidad a futuro.

La emulación, mejorada

- Las técnicas utilizadas para la emulación han mejorado tremendamente en los últimos diez años
- Los emuladores hacen hoy traducción predictiva y compilación del código a ejecutar a formatos nativos (traducción dinámica)
- Incluso guardan copias convertidas/compiladas del código a emular (Just In Time)
- 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 hace falta optimización
- ...Y las usamos transparentemente, todos los días.

Qemu: un caso muy especial

- `Bochs` es un emulador libre de x86; existe desde 1994, orientado a las estaciones de trabajo Unix.
- Bochs implementó un BIOS básico de PC, y la emulación de los principales dispositivos (discos, consola, VGA, puertos...)
- `Plex86` (originalmente FreeMWare, 1999, haciendo clara alusión a VMWare) ofrece una fuerte aceleración a Bochs, a través de la *traducción dinámica* permitiendo que en arquitectura x86 el código nativo no-peligroso corra directamente en el CPU sin pasar por emulación — y aislando/emulando las partes *peligrosas*.
- En 2003, `Qemu` retoma este trabajo y agrega el módulo de kernel `kqemu` (gratis, no-libre), atrapando estas llamadas con mucho mejor rendimiento (igualando el de VMWare).
- Qemu es la base para... lo que veremos a continuación :)

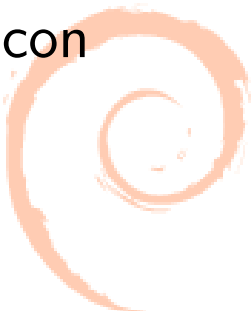
Temas

Virtualización asistida por hardware

- Buena parte del ruido que hoy en día está recibiendo la virtualización en general es específicamente por esta modalidad
- Algunas arquitecturas de cómputo —especialmente máquinas diseñadas para ser *grandes*— incluyen provisiones para ser *virtualizadas*
- El primer ejemplo es la IBM S/360-67 con el sistema CP-67/CMS (1968-1972). El CMS es un sistema operativo ligero, monousuario, pensado en que siempre existirían múltiples instancias en ejecución bajo el *hipervisor* CP
- Estas eran computadoras inherentemente de *tiempo compartido*, diseñadas para dar a sus diferentes usuarios la *ilusión* de tener una computadora dedicada a ellos

La motivación detrás de CP-67/CMS

- Las principales motivaciones para la virtualización son maximizar el aprovechamiento de recursos y proveer administración centralizada
- Al virtualizar el sistema completo, éste sistema ofrece mucha mayor aislamiento, seguridad y confiabilidad que cualquier sistema de tiempo compartido
- Permite además correr cualquier programa diseñado para una máquina S/360 — Incluso si éste no estaba diseñado para tiempo compartido
- IBM reimplementó este sistema como VM/370, al contar con una arquitectura con memoria virtual. z/VM, derivado de éste, sigue ampliamente en uso hoy en día.



CP/CMS y VM como software libre

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

El *hipervisor*: Más abajo que el kernel

¿Qué es un *hipervisor*?

- 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*
- Este micro-SO es conocido como el hipervisor (dando a entender que hace aún más que *supervisar*, el rol tradicional del SO)
- Idealmente, el kernel 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 haber múltiples niveles de hipervisores

El panorama hasta ≈ 2005

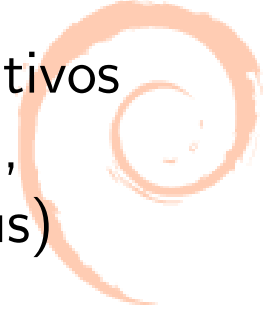
- Las arquitecturas que proveían virtualización por hardware eran muy especializadas — y por tanto, muy caras y fuera del alcance de los usuarios en general
- La virtualización por hardware estaba fuera del alcance de la mayor parte de los desarrolladores
- En 2005, Intel lanza la *Vanderpool Technology for IA-32 Processors (VT-x)*; en 2006, AMD lanza los procesadores con las extensiones *Pacifica*
- El tema era en verdad tan novedoso que tardó algunos años en desarrollar tracción

La virtualización asistida en nuestros sistemas

- Teniendo el hardware adecuado, este tipo de virtualización es la más sencilla de emplear
- Cada uno de nuestros hosts virtuales puede correr con una copia normal del sistema operativo que satisfaga nuestras necesidades
- Sí hay una pequeña penalización en el rendimiento global — Pero es típicamente despreciable

Estabilidad por virtualización

- Es aceptado universalmetnte que la mayor fuente de inestabilidad en un sistema operativo son los drivers
- Es código típicamente más *sucio* que el de otras partes del núcleo, y proviene de todo tipo de fuentes (desde desarrolladores independientes hasta las compañías desarrolladoras del hardware)
- Controlando 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
- Típicamente el hipervisor ofrecerá a los huéspedes dispositivos relativamente viejos y simples (p.ej. tarjetas de red NE2K, tarjetas de sonido SoundBlaster16, tarjetas de video Cirrus)



Volviendo a Kqemu: Emulando HVM

- KQemu fue liberado bajo la GPL en 2007
- KQemu agrega a una arquitectura que no lo implementa por hardware las funciones básicas de HVM. Claro, con una importante penalizacion en velocidad
- Sin embargo, como esto ocurre sólo cuando trabaja sobre código nativo, qemu+kqemu sigue ofreciendo una velocidad general muy aceptable

¿Qué proyectos/productos puedo usar?

- Xen (modo asistido)
- KVM (sobre Linux)
- Logical Domains (sobre Solaris)
- VirtualBox (base libre, con componentes propietarios)
- VMWare (no libre)
- Microsoft VirtualPC y HyperV (no libres)
- ...y muchos más

Temas

Un enfoque más ligero, más accesible

- Aún si la virtualización asistida ya está disponible en CPUs disponibles masivamente, no es parte de la mayor parte de sus modelos — Es una caracteristica *de lujo*
- La *paravirtualización* consiste en reescribir las porciones de un sistema operativo que interactúan directamente con el hardware, para que soliciten estas operaciones a un hipervisor
- Es también conocida como *Virtualización asistida por el OS* (OS-assisted virtualization)
- Formalmente podría verse como un *port* del sistema operativo a una nueva arquitectura — *Muy* parecida a la del sistema anfitrión

Paravirtualización: Apto para el Software Libre

- Requiere de 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 diferente arquitectura de paravirtualización
- El artículo con el cual se presentó Xen 1.x, habla de un port de Windows XP a su paravirtualizador, basado en el Academic Licensing Program de Microsoft — Pero no es redistribuible, sólo puede ser utilizado internamente en Xensource

Aprovechamiento de recursos en la paravirtualización

- 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 *idealizados*, no hace falta emular hardware real
- El OS huésped puede pedir al anfitrión recursos adicionales cuando los requiere (incluso sobre demanda — *balooning*)
- Puede haber un monitoreo mucho más completo — El OS anfitrión no tiene que saber tantos detalles del funcionamiento del huésped si éste se los confía

Hardware virtualizado, dispositivos paravirtualizados

- Este punto es empleado por todo tipo de virtualizadores — Paravirtualización a nivel dispositivo
- Entre más sencillos sean los dispositivos emulados para la virtualización, menos tiempo se perderá traduciendo llamadas a hardware inexistente
- Hasta una tarjeta de hace 10 años tiene hardware innecesario a la hora de virtualizar — entre más delgada sea la emulación, mejor rendimiento obtendremos
- Las clases de dispositivos `virtio` y `pv` llegan a ofrecer rendimiento de 5 a 10 veces mayor que la emulación de dispositivos reales (dependiendo de muchos factores)

¿Qué proyectos/productos puedo usar?

- Con mucho, la principal arquitectura libre de este tipo es Xen (2003).
- Del lado propietario, VMWare ofrece un modo de operación basado en la paravirtualización
- Hay otros proyectos en escena, como Denali (2001), pero se han mantenido principalmente como ejercicio académico

Xen y KVM: Dos caminos completamente distintos

- Los principales contendientes libres son Xen y KVM
- Sus ofrecimientos son en buena medida comparables — y es de esperarse que se produzca una lucha por la supervivencia, y uno de ellos termine quedándose con el pastel
- ...Yo pronostico que el ganador será KVM. Revisemos rápidamente la filosofía básica de ambos

Xen y KVM: Dos caminos completamente distintos

- Los principales contendientes libres son Xen y KVM
- Sus ofrecimientos son en buena medida comparables — y es de esperarse que se produzca una lucha por la supervivencia, y uno de ellos termine quedándose con el pastel
- ...Yo pronostico que el ganador será KVM. Revisemos rápidamente la filosofía básica de ambos

Xen: Un hipervisor mínimo

- Xen es un hipervisor puro — GRUB llama al núcleo de Xen, y éste lanza al kernel del sistema con el que interactuaremos
- El kernel tiene que estar compilado para correr dentro de la *arquitectura virtual* que nos ofrece Xen — No podemos usar un kernel estándar
- Esta primer máquina virtual se convierte en nuestro sistema supervisor — En idioma Xen, `Dom0`
- Todas las máquinas virtuales que lancemos son `DomU`
- Dom0 se comunica con Xen a través del demonio `xend`
- Hay reportes (no fáciles de reproducir) que hablan de inestabilidad de Xen en hardware de 64 bits bajo cargas pesadas de I/O

KVM: El Linux de siempre... mas un módulo medio raro

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

libvirt

- El proyecto libvirt busca ofrecer una biblioteca de abstracción capaz de manipular diferentes arquitecturas de virtualización
- Actualmente soporta Xen, QEmu, KVM, LXC, OpenVZ
- Nos promete migrar de una infraestructura a otra sin dolor
- Está orientado a simplificar la creación de interfaces de administración y monitoreo
- ...Pero su interfaz no es suficientemente rica, nos ofrece un subconjunto de la funcionalidad de cada una de ellas

Conclusiones de Xen vs. KVM

- Xen es un sistema maduro y probado, con herramientas muy completas y robustas
- ...Pero Xen es muy intrusivo — Requiere parches extensos al kernel. Desde 2.6.18 hasta 2.6.27, ningún otro kernel pudo ser Dom0.
- KVM es muy ligero, y fue aceptado ya como parte del árbol oficial de Linux
- ...Pero KVM es aún nuevo como virtualizador; carece de buenas herramientas de administración
- Xen no tiene el soporte completo de hardware de un núcleo como Linux - Específicamente la carencia de ACPI lo hacen inviable para muchos fines

Conclusiones de Xen vs. KVM

- Xen es un sistema maduro y probado, con herramientas muy completas y robustas
- ...Pero Xen es muy intrusivo — Requiere parches extensos al kernel. Desde 2.6.18 hasta 2.6.27, ningún otro kernel pudo ser Dom0.
- KVM es muy ligero, y fue aceptado ya como parte del árbol oficial de Linux
- ...Pero KVM es aún nuevo como virtualizador; carece de buenas herramientas de administración
- Xen no tiene el soporte completo de hardware de un núcleo como Linux - Específicamente la carencia de ACPI lo hacen inviable para muchos fines

¿No notan aquí la oportunidad de participar en algo grande? ;-)

Temas

Contenedores

¿Qué es un contenedor?

- Una diferente manera de virtualizar
- Más sutil
- Dentro del mismo núcleo
- Con mayores limitantes, pero con importantes ventajas

Orgullosos herederos del chroot

- Los sistemas Unix han ofrecido la llamada al sistema `chroot` desde 1982
- Bill Joy lo introdujo para facilitarle probar la construcción de nuevas versiones (preparando 4.2BSD) sin modificar sus fuentes
- Un proceso al que se le aplica chroot no puede ver el sistema de archivos fuera del directorio especificado
  - ...Bueno, no sin aplicar... Algunos trucos
- Chroot sólo afecta al árbol del directorio — No es (ni busca ser) es un verdadero aislamiento

¿No lo es? ¡Oportunidad para adecuarlo!

- Los contenedores construyen sobre de las llamadas 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*, 2002), Solaris 10 en adelante (*Zones*, 2005), y NetBSD/OpenBSD (*Sysjail* utilizando *Systrace*, 2006)
- ¿Sistemas no-libres? Parallels Virtuozzo (Linux, Windows), IBM Workload Partitions (AIX)

Principios básicos de los contenedores

- Llamamos a cada servidor virtual un *contexto de seguridad*
- 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 iptables
  - Dispositivos
  - Límites en consumo de recursos (RAM, CPU)
  - Algunos límites en ciertas llamadas al kernel
- Formalmente, los contenedores no implementan virtualización sino restricción

Variedad, con un muy pequeño límite

- A través de los contenedores, la virtualización es casi completa
- Se ven un poquito las *costuras* — Pero para propósitos prácticos, podemos ver a cada contenedor como un sistema independiente
- Excepto por el núcleo, podemos tener cualquier distribución corriendo dentro de nuestros contenedores al mismo tiempo
- La única restricción: Todos los contenedores corren con el mismo sistema operativo, con el mismo núcleo.
- Sysjail (OpenBSD) permite huéspedes *BSD y Linux a través de emulación de llamadas al sistema — Pero el núcleo *real* sigue siendo OpenBSD

Consumo de recursos óptimo

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

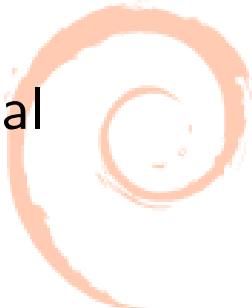
Temas

Traducción de APIs

- Otro tipo de virtualización, frecuentemente confundida con la emulación
- Consiste en implementar una capa que permita traducir una arquitectura *de software*, no de hardware
- Permite correr software diseñado para diferentes sistemas operativos simultáneamente, en el mismo OS, en la misma computadora
- El grado de integración, claro, depende de la similtud entre el sistema real y el... uhm... ¿huesped?

Entre Unixes

- Los diferentes Unixes libres implementan capas de traducción para poder ofrecer compatibilidad binaria para programas compilados para Unixes más populares
- En Linux, de 1994 a $\approx$ 1999 se mantuvo el subsistema iBCS, que ofrecía compatibilidad binaria con i386 BSD (386BSD, FreeBSD, NetBSD, BSDI/386), SVR3, SVR4 SCO, Wyse V/386, Xenix V/386, Xenix V/286... Pero se hizo obsoleto al volverse Linux más popular que todos ellos
- En los diferentes BSDs hay subsistemas de compatibildad, principalmente orientados hacia Linux
- En ambos casos, la integración de los binarios ejecutados al sistema huésped es prácticamente perfecta



Sistemas de diferente naturaleza

- Las traducciones de API más atractivas —pero también más difíciles— son las que ofrecen la integración de aplicaciones hechas para sistemas operativos muy distintos
- Por ejemplo, *WINE* (Wine Is Not an Emulator) para correr aplicaciones de Windows en Linux, o *CygWin*, *Microsoft Interix / SUA* (Subsystem for Unix Application) para correr aplicaciones que requieren un entorno POSIX en Windows
- La integración no *puede* ser perfecta — Se ven claramente las costuras en las muy diferentes metáforas ofrecidas por ambos sistemas.
- La compatibilidad no es completa, y hay muchas aplicaciones que no funcionan o se rompen.

Temas

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), en sistemas embebidos y en equipos tradicionales

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), en sistemas embebidos y en equipos tradicionales

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), en sistemas embebidos y en equipos tradicionales

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), en sistemas embebidos y en equipos tradicionales

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), en sistemas embebidos y en equipos tradicionales

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), en sistemas embebidos y en equipos tradicionales

Diferentes necesidades, diferentes soluciones

- Hay una gran riqueza en la oferta de herramientas de virtualizacón
- Cada herramienta, cada estrategia tiene características muy distintas
- Pero lo más importante: Hasta hace muy poco, estaba completamente fuera del radar. Nos ofrece soluciones muy interesantes a una amplia categoría de problemas
- Además, es un área muy interesante donde podemos participar
- La virtualización es un componente complejo y dificil de probar a fondo para las distribuciones. Una buena manera de contribuir es participando en esta área.

Gracias

Para armar esta presentación, debo agradecer a:

- Jan Lübbe, por brindarme un magnífico documento en que basarme y por su trabajo en `http://wiki.debian.org/SystemVirtualization`
- Mis compañeros de trabajo, Aristeo Tovías y Patricia Llanas, por sufrir todos los días los embates de los virus en Windows y obligarme a empaparme en este tema ;-)
- Rolando Cedillo, gurú redhatero mexicano, por empujarme por esta senda — y por permitirme robar algunas ideas de su ponencia minutos antes de la mía ;-)

Gracias

Para armar esta presentación, debo agradecer a:

- Jan Lübbe, por brindarme un magnífico documento en que basarme y por su trabajo en `http://wiki.debian.org/SystemVirtualization`
- Mis compañeros de trabajo, Aristeo Tovías y Patricia Llanas, por sufrir todos los días los embates de los virus en Windows y obligarme a empaparme en este tema ;-)
- Rolando Cedillo, gurú redhatero mexicano, por empujarme por esta senda — y por permitirme robar algunas ideas de su ponencia minutos antes de la mía ;-)

¿Dudas?

¡Gracias!

Gunnar Wolf — gwolf@gwolf.org
http://www.gwolf.org/soft/virt