La virtualisation vue de l'intérieur

Introduction au fonctionnement des machines virtuelles

Damien Dejean

5 février 2013

Plan

Bref historique

70's	Prémices de la virtualisation (IBM Cambridge center/MIT), premier support hardware par IBM.
1974	Gerald J. Popek et Robert P. Goldberg définissent les pré-requis pour la virtualisation.
1999	VMware ESX et Workstation.
2003	Xen.
2005	Intel VT-x.
2006	AMD-v.
2007	VirtualBox par InnoTech (acquisition par Sun en 2008).
2008	KVM (acquisition par RedHat).

Quelques questions

- Selon vous, Qu'est-ce qu'une machine virtuelle ?
- Quelle est la différence entre émulation et virtualisation ?

Plan

Plan

Ce que l'on voit d'une machine virtuelle

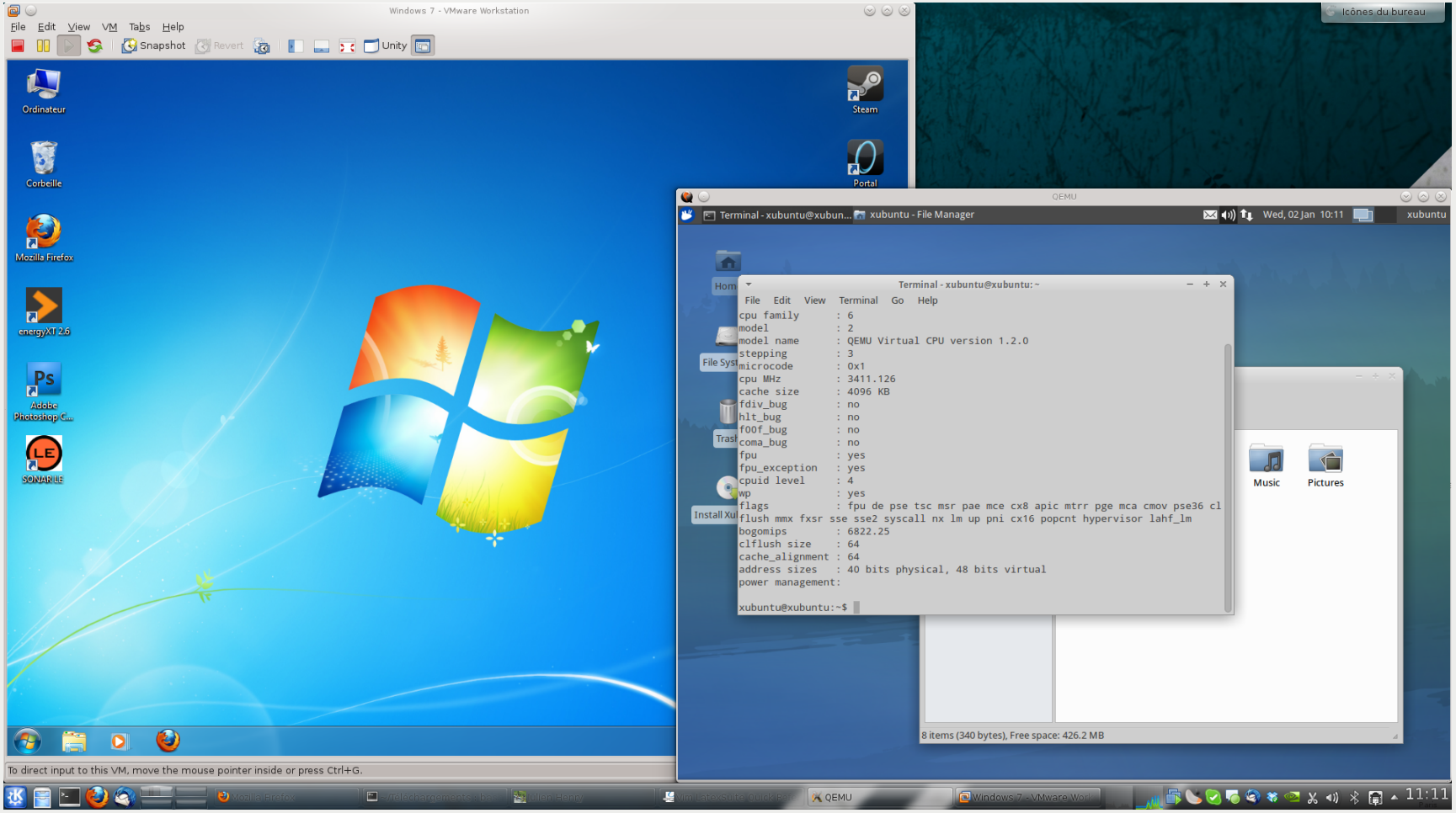


Figure : Deux machines virtuelles sur un environnement GNU/Linux

Définition formelle

Popeck et Goldberg ont établi des critères pour qualifier un hyperviseur :

Equivalence les programmes doivent avoir un comportement similaire sur machine réelle ou virtuelle.

Efficacité une majorité des instructions doivent s'exécuter directement sur le CPU.

Contrôle des ressources le programme virtualisé ne doit avoir accès aux ressources qu'à travers le moniteur, qui peut reprendre la main à tout moment.

Classification des hyperviseurs

Les hyperviseurs sont divisés en deux classes indiquant leur mode de fonctionnement :

Type 1 (bare-metal) fonctionne directement sur machine nue, assurant l'interface entre les OS virtualisés et la machine.

Type 2 (hosted) fonctionne par-dessus un système d'exploitation existant, une machine virtuelle se présente alors comme un processus. L'hyperviseur doit intégrer une partie de ses fonctions au noyau par-dessus lequel il fonctionne.

Hyperviseur de type 1

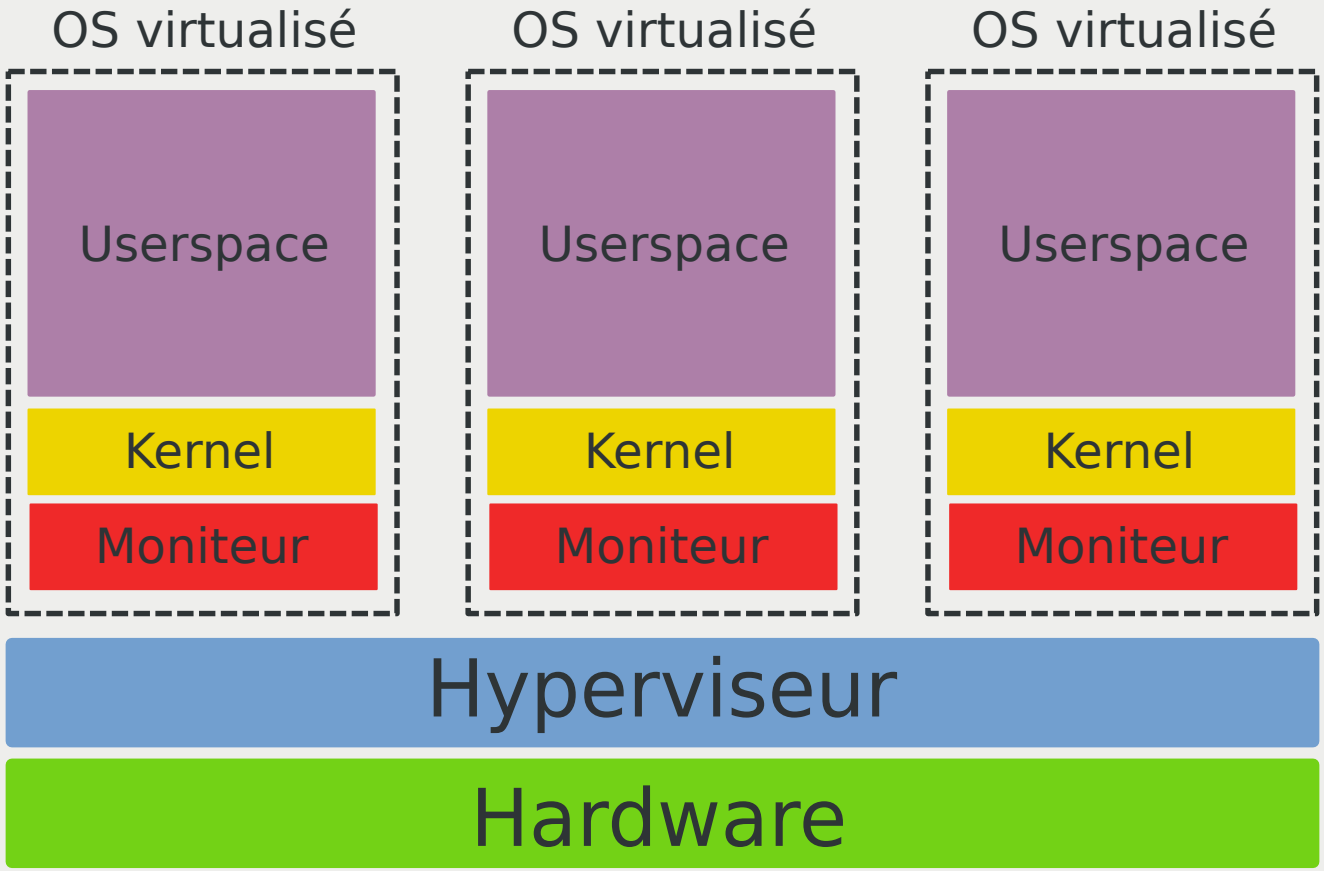


Figure : Représentation d'un hyperviseur de type 1

Exemples : Xen, VMware ESX, Microsoft Hyper-V, Oracle VM Server.

Hyperviseur de type 2

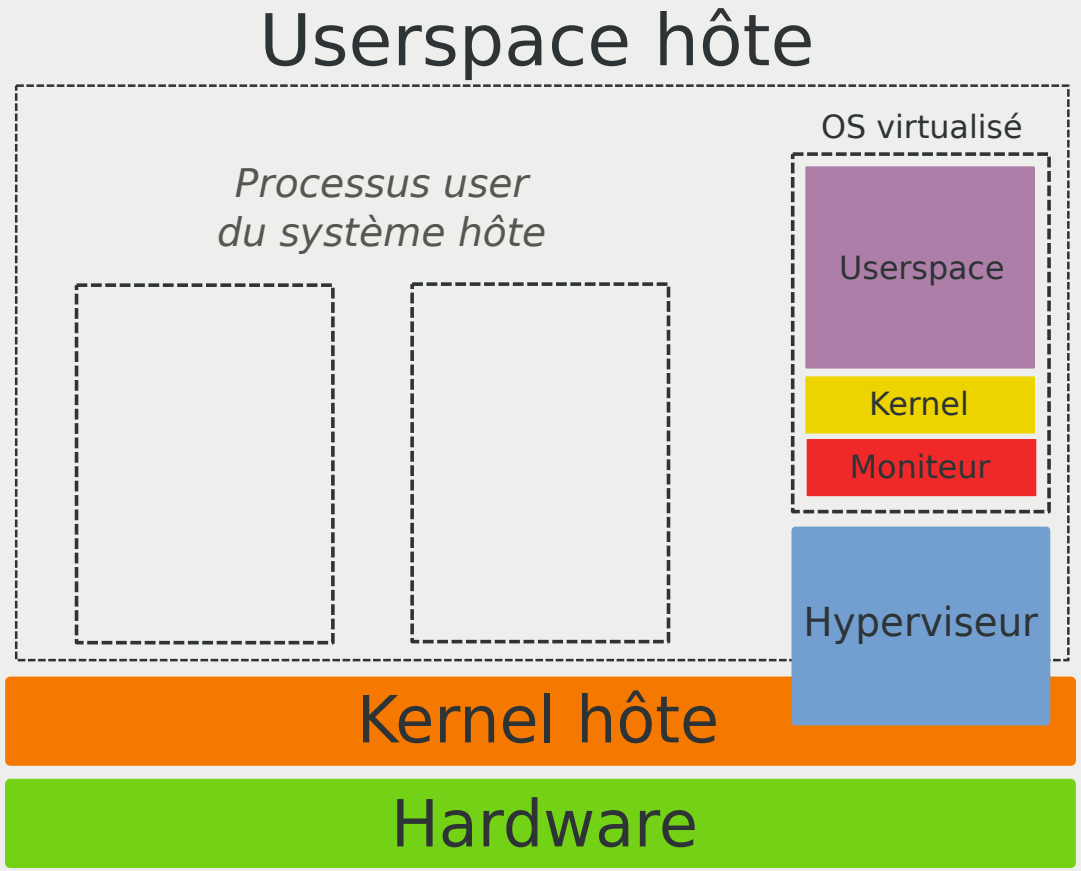


Figure : Représentation d'un hyperviseur de type 2

Exemples : QEMU/KVM, Oracle VirtualBox, VMware Workstation.

Plan

Plan

Le minimum syndical

Les pré-requis pour pouvoir se salir les mains :

- interruptions/exceptions
- niveaux de privilèges d'un CPU
- kernel/userspace
- mémoire virtuelle

Principe général

Que se passe-t-il si l'on tente d'exécuter un kernel comme un processus classique ?
Erreurs les plus probables :

- faute de protection (*General Protection Fault* sur x86)
- faute de page (*Page Fault* sur x86)
- et d'autres...

Problème : l'exécution d'un OS dans un contexte utilisateur va échouer chaque fois qu'un accès ou une instruction requérant un niveau de privilège élevé se produit.

Solution

Un accès ou une instruction nécessitant un niveau de privilège élevé va provoquer une exception (une faute) si elle est réalisée en mode non privilégié. La solution consiste à rattraper les fautes commises par le système que l'on veut virtualisé.

1. Exécution de l'instruction problématique
2. Exception
3. Traitement de l'exception par émulation de son comportement
4. Instruction suivante

Conséquences

L'émulation du code privilégié n'est pas sans conséquences :

- l'exécution du code noyau est globalement plus lente que l'exécution de code user
- une instruction ayant un comportement particulier en mode superviseur doit impérativement "trapper" quand elle est exécutée en mode user

Problème : certaines instructions privilégiées peuvent s'exécuter sans provoquer de fautes.

Jeux d'instructions non virtualisables

Les architectures actuelles (x86/x86_64, PPC, ARM,...) présentent des instructions qui peuvent s'exécuter indifféremment en mode kernel ou user, et qui ont un comportement différent selon le mode.
Pas d'exceptions, pas de virtualisation.
Solutions :

- paravirtualisation du noyau
- binary translation
- support matériel

Solution 1 - Paravirtualisation du noyau

Remplacer dans le code du noyau, ou le binaire généré, les instructions problématiques par un op-code qui :

- provoque une exception qui puisse être rattrapée (ex : *invalid opcode*)
- identifie l'instruction à émuler et ses paramètres

Chacune de ces instructions est rattrapée par le moniteur du fait de l'exception et peut être émulée.

Avantages

- performance
- simplicité

Inconvénients

- nécessite les sources
- coûteux en développement (analyse de code) et limité en portée

Solution 2 - Binary translation

C'est la paravirtualisation automatique du code exécuté. Le code est lu à l'avance et les instructions problématiques sont remplacées par les op-codes magiques.

Avantages

- polyvalence : n'importe quel système peut être exécuté
- le code généré peut utiliser des instructions privilégiées (invalidation cache, tlb...)

Inconvénients

- performances
- complexité (nécessité de caches)

Solution 3 - Support matériel

Le matériel apporte la solution au problème. Une instruction nécessitant un niveau de privilège élevé trappe systématiquement. C'est le support hardware de *première génération*. Avec le support des niveaux d'exécution, certaines instructions n'ont plus besoin de trapper.

Avantages

- polyvalence : n'importe quel système peut être exécuté
- performance
- réduction des coûts de maintenance du code (full-virtualization)

Inconvénients

- pas disponible sur toutes les architectures et tous les modèles de processeurs

Et maintenant ?

On sait...

- rattraper les instructions privilégiées
- traiter les instructions problématiques

Il reste à...

- fournir de la mémoire au système virtualisé
- proposer des périphériques

Situation initiale

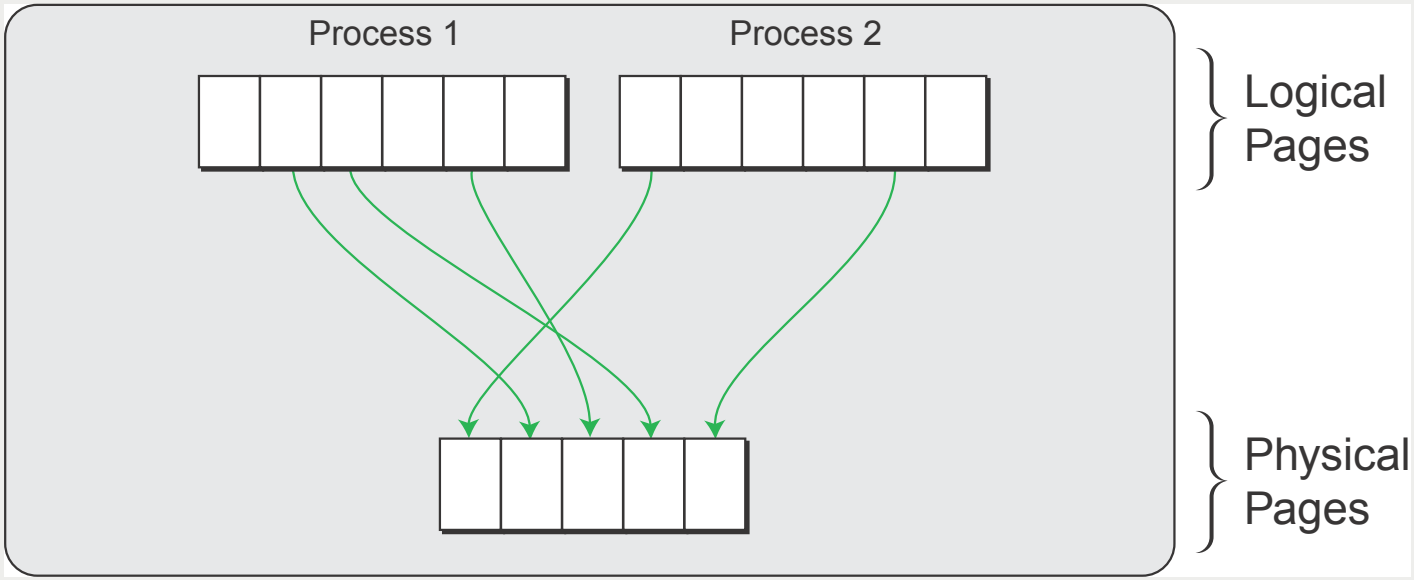


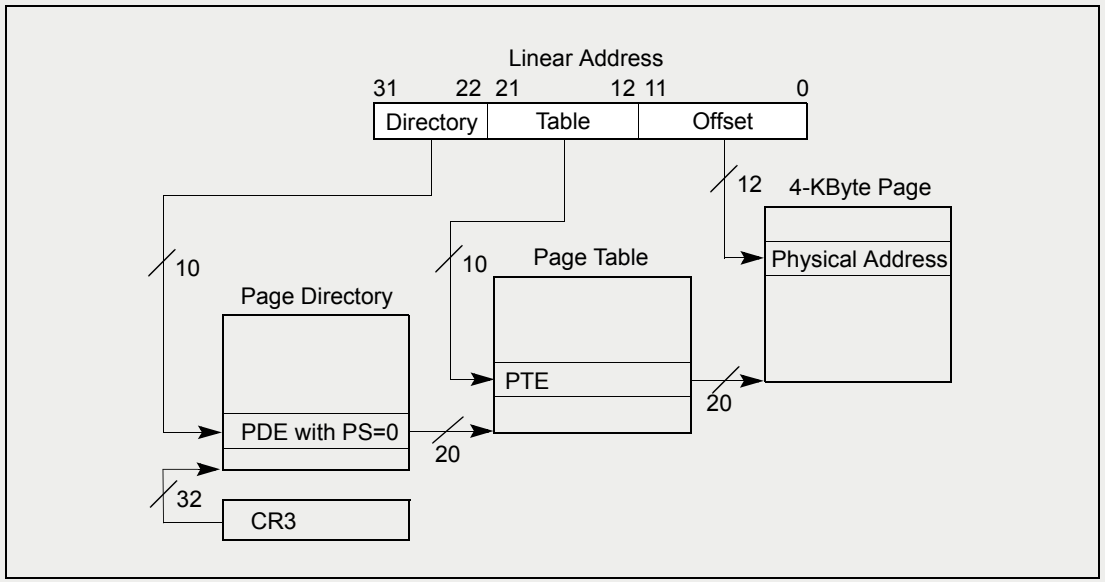
Figure : Native System Memory Management Unit[1].

- un espace d'adressage physique
- maintient d'un mapping mémoire virtuelle/mémoire physique en utilisant une table des pages (en vert)
- une MMU *(Memory Management Unit)* chargée de la traduction

Illustration

Résolution d'une adresse virtuelle en adresse physique :

1. Un processus accède une adresse de son espace virtuel
2. La MMU parcourt la table des pages et résout l'adresse virtuelle en adresse physique
3. La résolution est placée dans un cache local appelé *Translation Lookaside Buffer* (TLB) pour réutilisation
4. Le CPU émet l'adresse physique sur le bus mémoire et fournit la donnée au processus



Support logiciel : Shadow Page Table

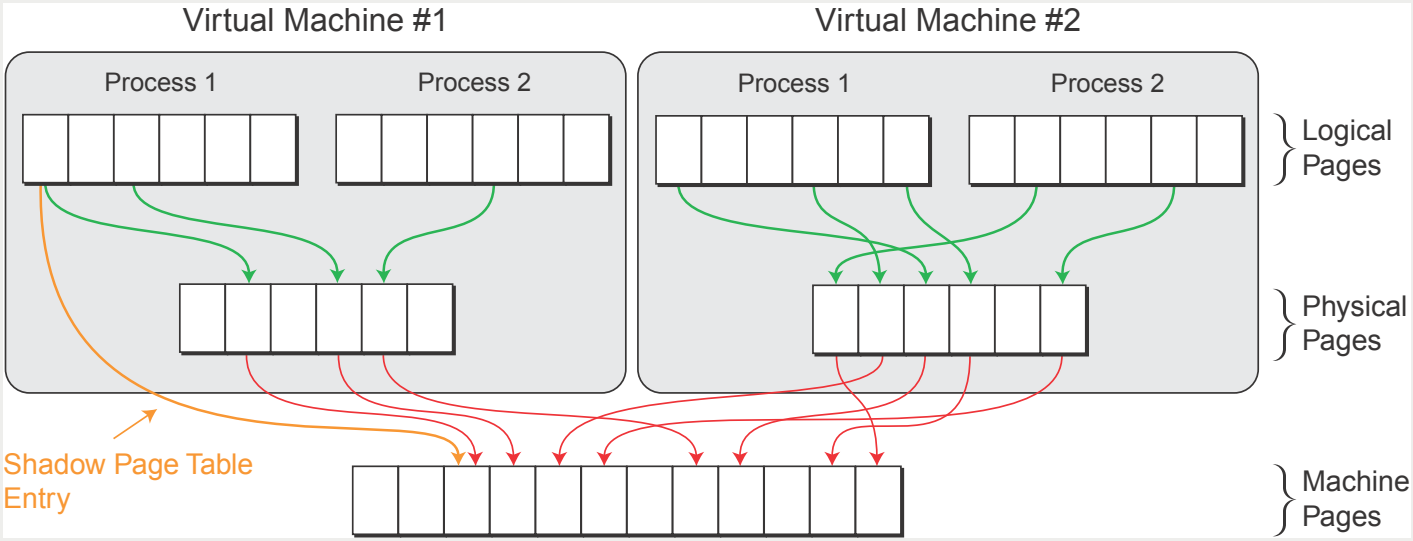


Figure : Shadow Page Tables[1].

1. Le noyau maintient sa table des pages *(en vert)*
2. Le moniteur connaît le mapping entre l'espace d'adressage physique du guest et la mémoire du host *(en rouge)*
3. Le moniteur maintient la vraie table des pages du CPU avec le mapping "virtuel guest" vers "physique host" *(en orange)*.

Solution logicielle

Ce mécanisme est appelé "Shadow Page Tables" et est le mécanisme utilisé dans les hyperviseurs qui n'ont pas de support matériel pour la virtualisation de la MMU.

Avantage

- Traduction d'adresse à coût identique au host (un seul parcours de table des pages)

Inconvénients

- Coût élevé d'une faute de page
- Gestion des niveaux de privilèges compliquée

Solution matérielle

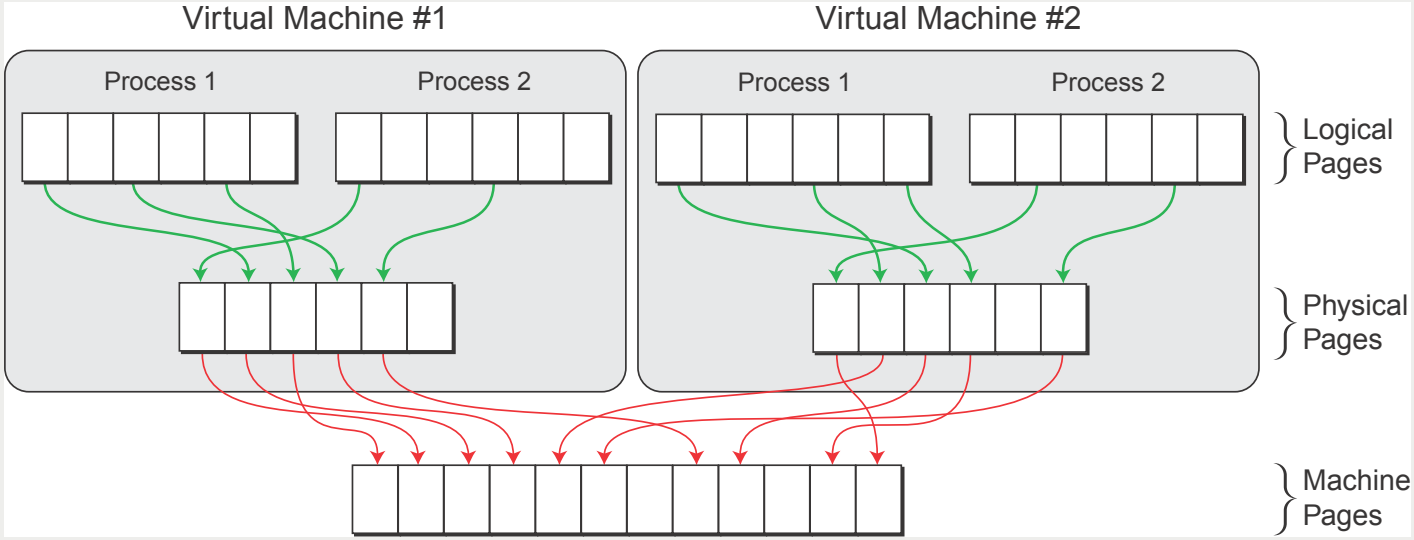


Figure : Hardware Memory Management Unit Diagram[1].

- Le noyau maintient ses tables des pages pour mapper ses processus *(en vert)*.
- Le moniteur maintient une table des pages contenant le mapping des "adresses physiques guest" vers les "adresses physiques host" *(en rouge)*.

Solution matérielle

Cette technique est celle proposée par le support matériel de la virtualisation de seconde génération.

Intel	*Extended Page Table (EPT)*
AMD	*Rapid Virtualization Indexing (RVI)* ou *Nested Pagination*
ARM	fait partie de *Virtualization Extension (VE)*

Avantages

- MMU dédiée, facilité de gestion
- Transparence des niveaux de privilèges

Inconvénients

- Double coût de lecture de la table des pages
- Matériel compatible

Équilibrage mémoire

Problématique

Une machine virtuelle a un espace d'adressage réservé qu'elle peut utiliser à tout moment. Cette mémoire n'est donc pas disponible pour le host.

Figure : Situation mémoire sans ballooning

Équilibrage mémoire

Solution : *le ballooning*

Mécanisme (processus, module noyau) introduit dans le guest qui a pour fonction :

- d'allouer de la mémoire dans le guest, pour la rendre au host
- libérer cette mémoire quand le guest en a besoin

Figure : Mémoire allouée par le balloon, rendue au host.

L'ensemble est réalisé sous contrôle de l'hyperviseur en fonction des besoins du host et du (des) guest(s).

Et maintenant ?

On sait...

- rattraper les instructions privilégiées
- traiter les instructions problématiques
- fournir de la mémoire au système virtualisé

Il reste à...

- proposer des périphériques

Matériel d'une VM

Une machine virtuelle nécessite d'autres périphériques qu'un simple couple processeur/mémoire. Ce qui pose un certain nombre de questions :

- Peut-on laisser le guest accéder directement au matériel ?
- Lesquels ?
- Sur quels types d'hyperviseurs ?
- Les performances ?
- Quelles sont les solutions disponibles ?

Matériel d'une VM

L'objectif est d'arriver à présenter du matériel à la machine virtuelle. Il existe deux approches pour la virtualisation du matériel :

full virtualization le moniteur émule le comportement d'un matériel, ou permet un accès direct au matériel de la machine physique (exclusivité)

paravirtualisation le guest est modifié, sur la base de drivers en général, pour communiquer directement avec le moniteur

Exemple à suivre : application sur une pile OpenGL, une manière de fournir l'accélération graphique à une machine virtuelle.

Illustration : une pile OpenGL

Comment virtualise-t-on cette pile ? Comment fournir un support GPU au système d'exploitation invité ?

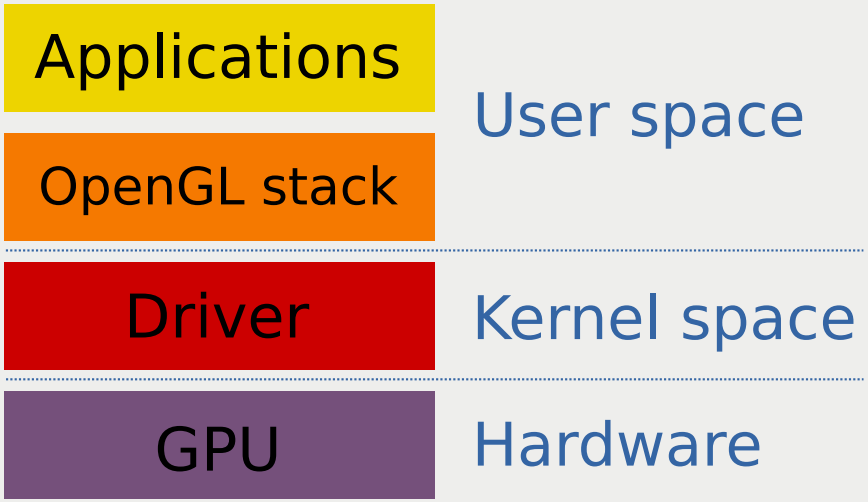


Figure : Pile OpenGL sur un système classique.

- un vendeur fournit (normalement) le GPU, le driver et la bibliothèque, ils sont indissociables
- les applications chargent la bibliothèque pour parler au GPU

Full virtualization

Le moniteur émule le matériel, la machine virtuelle utilise ses drivers classiques pour communiquer avec.

Exemples :

- QEMU émule une carte graphique Cirrus Logic GD5446 PCI, une carte réseau Realtek RTL8139 PCI.
- VirtualBox émule un adaptateur réseau Intel Pro/1000.

Le moniteur va intercepter la configuration et les sorties mémoire vers le matériel et émuler le comportement de celui-ci.

Full virtualization

Avantages

- Le matériel n'est pas voué à changer.
- Le guest reste inchangé, on utilise un driver existant, comme sur machine physique.

Inconvénients

- Peut être très complexe selon le matériel à émuler, parfois des milliers de registres, du bytecode à interpréter, etc.

Full virtualization

Avec notre pile OpenGL : driver et pile OpenGL d'un GPU du marché.

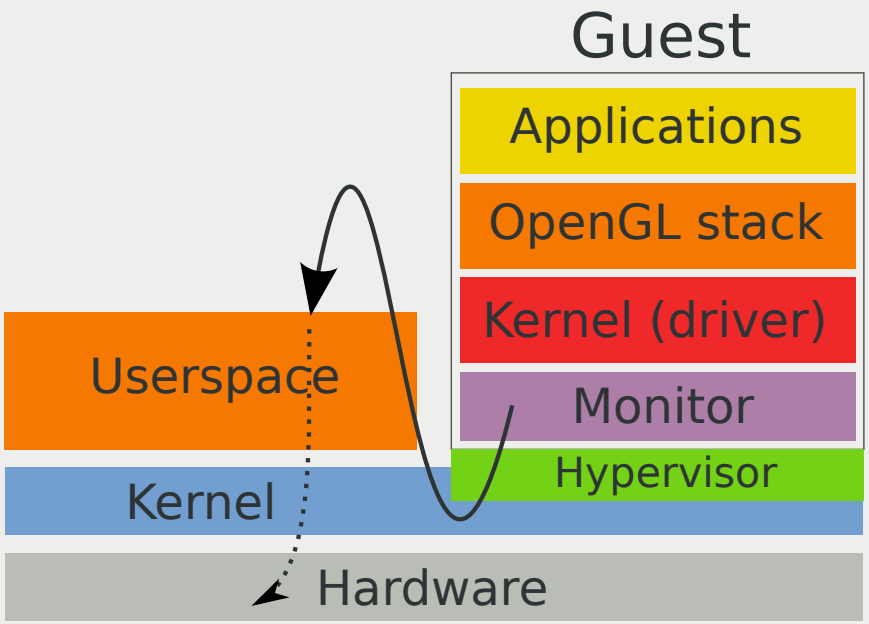


Figure : Full virtualisation d'une pile OpenGL.

1. Une application émet une commande OpenGL.
2. La commande est transformée en commandes GPU par la pile puis le drivers.
3. Le moniteur intercepte ces commandes et les renvoie dans le host pour traitement.
4. Un processus host est chargé de les traduire pour réaliser les équivalents OpenGL ou de parler directement au GPU.

Full virtualization

En pratique, la réalisation d'un driver 3D accéléré sous cette forme est trop complexe.

- La conversion de commandes GPU venant du guest vers le GPU du host est trop complexe sauf si les GPUs sont identiques.
- Si les GPUs sont identiques il faut un mécanisme de sauvegarde de contexte et de fenêtrage.
- Si l'on veut recréer des commandes OpenGL à partir de commandes GPU, il faut pouvoir faire le travail inverse du driver guest, trop complexe.

La seule alternative possible est la paravirtualisation.

paravirtualisation

Le guest est équipé de drivers ou de bibliothèques, qui viennent remplacer ceux du système par défaut. Le système d'exploitation est alors conscient qu'il est virtualisé.

Exemples :

Oracle VirtualBox "Guest Additions", fournissent l'intégration de la souris, un affichage accéléré matériellement, presse papier partagé, etc.

VMware Workstation "VMware Tools", fournissent l'accélération hardware, le glisser/déposer, etc.

Ces outils vont permettre de virtualiser un périphérique à plus haut niveau qu'en émulant le matériel.

paravirtualisation

Avantages

- Simplicité rapport à l'émulation de matériel.
- Meilleure intégration entre host et guest.
- Performances accrues avec certains périphériques.

Inconvénients

- Le guest est conscient de son état virtualisé.
- Tous les composants d'un guest ne sont pas paravirtualisables.

paravirtualisation

Application à notre pile OpenGL :

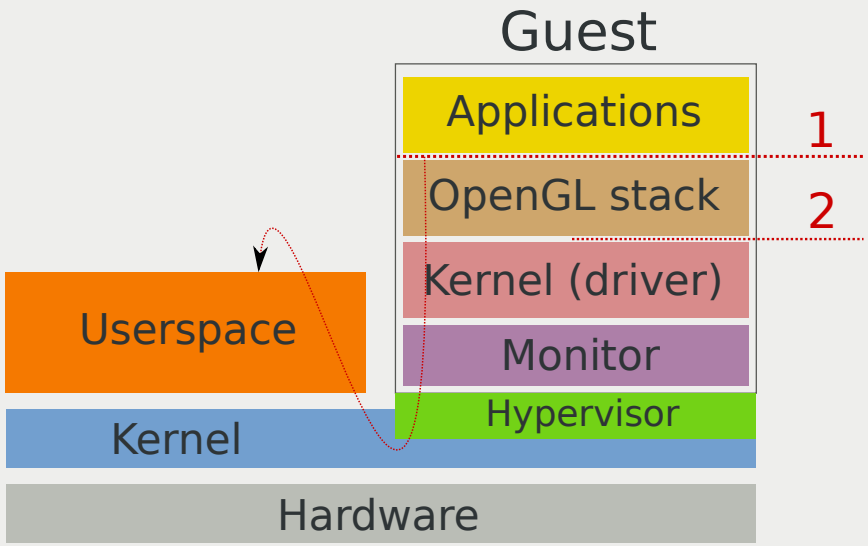


Figure : paravirtualisation d'une pile OpenGL.

Plusieurs choix possibles

1. Niveau API : commandes OpenGL transmises directement au host.
2. Niveau Driver : on garde une pile OpenGL, read/write/ioctl transmis au host, et interprétés.

Exemples : VMGL (XenGL), Émulateur Android (BlueStack)
Nécessité d'avoir un mécanisme de communication entre host et guest : réseau ou mémoire partagée par exemple.

Plan

Conclusion

En bref...

- mécanisme de base de la virtualisation
- virtualisation $\neq$ émulation
- pas de solution miraculeuse : intrusif et consommateur
- un bon niveau de performances nécessite un support hardware

Mais ça marche !

Questions ?

Remerciements

- *La Guilde* pour l'organisation, le suivi, les discussions.
- *Cyprien Laplace* pour les réponses à mes innombrables questions, la revue des slides.
- *Julien Henry, Tristan Braud et Jean Baptiste Guet* pour les conseils et la revue de ces slides.

Plan

Références I

Performance Evaluation of AMD RVI Hardware Assist
www.vmware.com/pdf/RVI_performance.pdf
VMware, 11 Mars 2009.

Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A : System Programming Guide, Part 1.
www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

Intel, Janvier 2013.

KVM - Kernel Based Virtual Machine
www.redhat.com/resourcelibrary/whitepapers/doc-kvm
RedHat, Septembre 2008.

Références II

La Virtualisation

student.ulb.ac.be/ fsanty/cours/ba3/rechcom/virtualisation.pdf

François Santy, Université Libre de Bruxelles, 2009-2010.