

Séance 5

Virtualisation de la machine



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

Objectifs

- Comprendre la notion de **machine virtuelle**
 - Machines virtuelles processus ou système
 - Description et rôle d'un Virtual Machine Monitor
- **Trois principaux types** de machines virtuelles
 - Virtualisation et interprétation par un hôte
 - Paravirtualisation et hyperviseur
 - Conteneurisation et virtualisation dans l'OS

Machine virtuelle



Machine virtuelle

- Exécution simultanée de plusieurs OS ou sessions d'OS

Sur une unique machine, que ce soit un PC ou un serveur

- Virtualisation par un moniteur de VMs ou hyperviseur

- Exécution de plusieurs applications sur plusieurs OS
- Chaque machine virtuelle a ses propres caractéristiques

- Machine virtuelle expose du hardware virtuel

Une instance d'un OS invité sur un système virtualisé

Machine virtuelle processus

- Application normale exécutée à l'intérieur d'un OS

Permet de s'abstraire des détails de l'hardware sous-jacent

- Environnement indépendant de la plateforme

Utilisé pour l'exécution d'un simple processus (programme)

- Typiquement implémenté comme un interpréteur

Par exemple la machine virtuelle Java (JVM)

Machine virtuelle système

- **Multiplexage** de l'hardware sous-jacent entre plusieurs OS

Permet de faire du partage de temps des ressources hardware

- Capable de faire tourner **plusieurs OS** différents

Partage des ressources hardware entre les VMs

- Peut être vu comme un **ordonnanceur** d'OS

Jeu d'instructions de la VM peut être différent du réel

Idée générale (1)

- Abstraction entre le hardware et le système d'exploitation

Ou sous un software qui accède directement au hardware

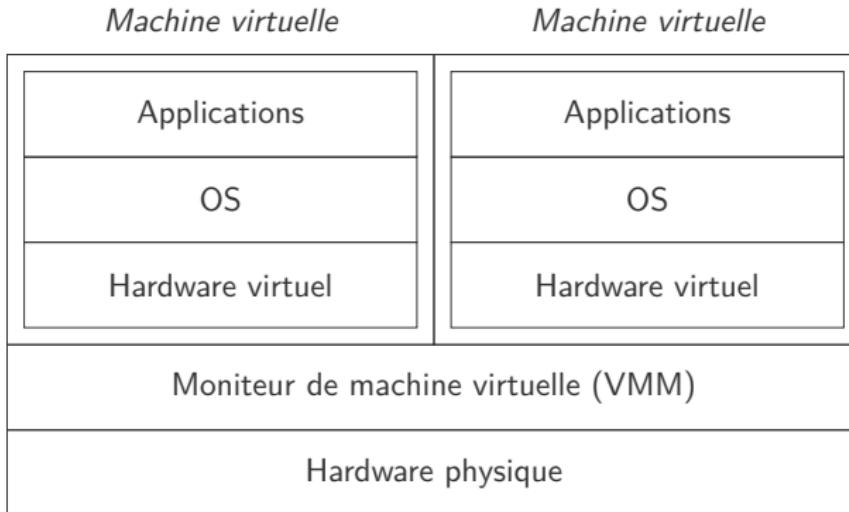
- Cette couche fait communiquer les deux acteurs

- Exposer du hardware virtuel au système d'exploitation
- Mapper l'utilisation de l'hardware virtuel sur le réel

- Contrôle par le moniteur de machine virtuelle (VMM)

Donne l'illusion de machines virtuelles isolées

Idée générale (2)



VMM vs OS

- VMM et OS placés par **dessus le hardware**
 - Isolation des parties prenantes (VM ou processus)
 - Attribution équitable des ressources hardware physique
- L'**OS** fournit un haut niveau d'abstraction
 - CPU et mémoire virtuelle, processus, fichier pour périphérique*
- Le **VMM** fournit un bas niveau d'abstraction
 - Sensation d'être sur son propre hardware avec accès direct*

Design d'un VMM

- Environnement identique au hardware physique sous-jacent

Le hardware virtuel exposé par le VMM dépend du sous-jacent

- Instructions de l'OS hôte transmise au hardware physique

La plupart du temps sans aucune modification par le VMM

- Gestion des ressources prise en main par le VMM

Tout ce qui n'est pas CPU : mémoire et périphériques

Acteurs

- Contrôle du hardware par le **système hôte**

Système tout à fait classique sur lequel tourne un OS

- Exécution d'un **système invité**/virtualisé

Repose sur le système hôte pour fonctionner

- Contrôle de la virtualisation par un **hyperviseur**

Assure la création et l'exécution de machines virtuelles

Types de virtualisation

- **Virtualisation** complète ou paravirtualisation

Selon que l'invité sait ou non qu'il est virtualisé

- Simulation complète d'un ordinateur par **émulation**

Même le microprocesseur est simulé

- **Container** permettant un environnement d'exécution isolé

Plusieurs instances user-space dans un même kernel

Pourquoi ?

- **Multiplexage** de l'hardware physique
Par exemple, datacenter multiplexe entre plusieurs clients
- Exécution de contenu dangereux et **isolation** des autres
Entre VMs, ou sur une machine pour ouvrir une pièce jointe
- Augmente la **productivité** des développeurs
Offre un environnement de développement « bac à sable »
- **Compatibilité arrière** avec d'anciennes architectures
Pour émuler des vieilles consoles de jeu, par exemple

Challenges de la virtualisation

- Gestion des **instructions privilégiées** envoyée par l'OS invité
- Exécution aussi **performante** que possible

Limiter au maximum l'overhead ajouté par le VMM

- Gestion de la **mémoire** à partager entre les VMs

Gestion efficace de plusieurs espaces d'adresses

- Virtualisation des **périphériques d'entrée/sortie**

Gestion des requêtes E/S provenant de plusieurs OS

Virtualisation



Interprétation par un hôte (1)

- VMM exécuté comme une application sur un **OS hôte**
 - Représentation du hardware physique au niveau logiciel
 - Exécution du code des VMs et mise à jour du hardware virtuel
- Hyperviseur de **type 2** qui travaille sur un OS
 - Hosted VMM*
- Plusieurs **logiciels existants** édités par des grosses firmes
 - Oracle VirtualBox, VMWare Fusion/Player/Server/Workstation, Parallels Desktop/Server, Microsoft VirtualPC/VirtualServer...*

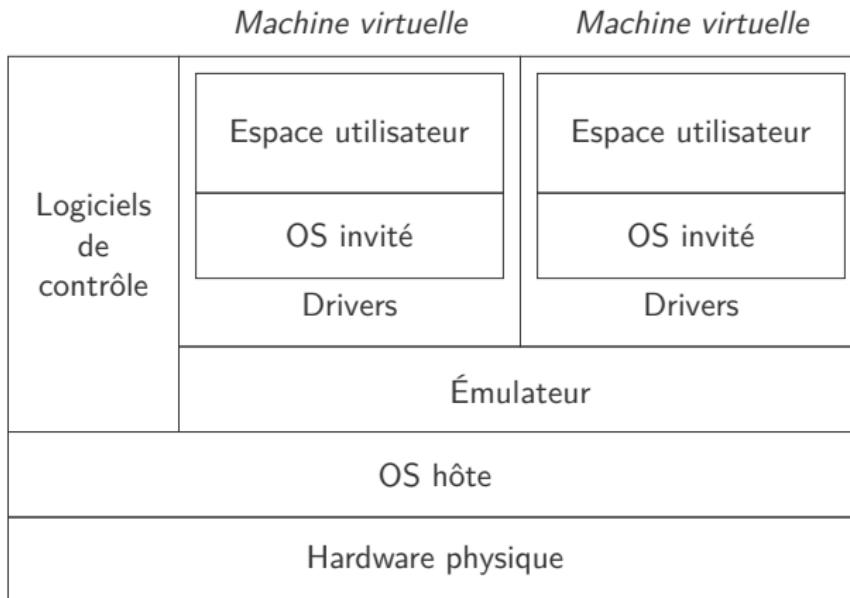
Émulation des programmes

■ Émulation logicielle du hardware physique

Exécution d'un programme qui maintient des structures

```
1 while (1)
2 {
3     instr = fetch (vm.PC);
4     vm.PC += 4;
5     switch (instr)
6     {
7         case ADD:
8             int sum = vm.regs[instr.reg0] + vm.regs[instr.reg1];
9             vm.regs[instr.reg0] = sum;
10            break;
11
12         case SUB:
13             // ...
14     }
15 }
```

Interprétation par un hôte (2)



Interprétation par un hôte (3)

- Offre une **isolation complète** des VMs
 - Gestion des instructions privilégiées (registre spécial, MMU...)
 - Contrôle total du hardware (blocage du réseau par exemple)
 - L'OS invité ne sait pas du tout qu'il est virtualisé
- L'**interprétation** de code est (très) lente

Émuler complètement un processeur moderne est difficile

Niveau de privilège

- Quatre **niveaux de privilège** sur x86 (rings)
 - Ring i peut lire et écrire mémoire Ring j ($j \geq i$)
 - Appel de fonctions entre Ring par mécanisme hardware
 - Ring 0 seul à pouvoir exécuter instructions privilégiées
- **Organisation typique** d'un système
 - L'OS est exécuté à l'intérieur du Ring 0
 - Les applications utilisateurs sont exécutées dans le Ring 3

Jeu d'instructions virtualisable

- Deux **caractéristiques** des instructions
 - Sensible : comportement différent (mode/config du processeur)
 - Privilégiée : provoque un trap vers mode privilégié
- Virtualisation de l'architecture **IA-32** (x86)
 - $\neg S, \neg P$: exécution directe par le processeur
 - S, P : VM en mode utilisateur, traps gérés par VMM
 - $S, \neg P$: 17 instructions délicates à gérer, VMM les interdit
- Le VMM peut s'interposer si le processeur est **virtualisable**

Contrôle de l'interaction des VMs avec le « monde extérieur »

Trap et émulation

- Si même jeu d'**instructions** entre invité et hôte
 - Ring 3 : Applications invitées
 - Ring 1 : OS invité
 - Ring 0 : VMM
- Trap vers le VMM lors d'**instructions privilégiées**
Émuler l'instruction, tuer la VM...
- Nécessite que le processeur soit **virtualisable**
Toutes les instructions sensibles doivent être privilégiées

Instruction push

- **Instruction push** place un registre sur la pile

Et registre %cs contient deux bits avec niveau privilège

- OS invité dans le Ring 1 peut **exécuter push %cs**
 - L'OS invité découvre qu'il n'est pas dans le Ring 0 !
 - Devrait générer un trap pour que le VMM place 0 sur la pile

Translation binaire

- Réécriture dynamique des instructions non virtualisables

Pour faire en sorte que le VMM soit appelé

- Les applications invitées ne doivent pas être réécrites

Seules les instructions non virtualisables sont traduites

- L'implémentation du VMM est délicat

Par exemple pour processeur avec un TLB niveau software

Processeur virtualisable avec TLB software

App invité (Ring 3)	OS invité (Ring 1)	VMM (Ring 0)
Accès mémoire avec TLB miss provoque un trap		Appel du gestionnaire TLB de l'OS invité
	N° de page virtuel depuis l'adresse virtuelle, cherche dans table des pages, récupère n° de cadre physique et mise à jour TLB	
	Instruction « return from trap »	Installe plutôt un mapping n° page virtuel – n° de cadre machine
Accès mémoire précédemment fautif réussit maintenant		Redémarrage de l'app invité

Support hardware

- Ajout de **support pour la virtualisation** dans le hardware
- Par exemple, **Intel VT-x**
 - Deux nouveaux modes d'exécution (root/non-root)
 - Instruction sensible en non-root provoque transition en root

Virtualisation du MMU

- Quatre différents **types d'adresses** à gérer
 - HPA/HVA : Host Physical/Virtual Address
 - GPA/GVA : Guest Physical/Virtual Address
- Le VMM doit maintenir une **shadow page table**

Si le hardware ne supporte qu'une seule table des pages
- Lourdeur des **changements de contexte** pour accès mémoire

Entre la VM et le VMM

Émulateur (1)

- Logiciel compilé pour un système exécuté sur un autre

Émulateur fournit une interface vers le système hôte

- Émulation de la totalité d'un système

CPU, sous-système mémoire, périphériques E/S

- Plusieurs logiciels d'émulation existants

QEmu, sys161 (MIPS), Bochs (x86)...

Émulateur (2)

- Permet l'exécution de **binaires compilés** pour un autre système
 - Protège de l'accès direct aux ressources de l'hôte
 - Émulation de périphériques n'existant plus
- Peut être **très très lent** à exécuter le programme émulé

Dépend de la complexité du jeu d'instructions à émuler

Wine

- On peut se limiter à réimplémenter l'**interface OS**
- **Wine** réimplémente l'API Windows
 - Exécution de binaire Windows non modifié
 - Appels à l'API mappé à des équivalent Linux/Solaris/MacOS...
- **Recompilation** d'applications Windows en native

Genre de « virtualisation d'API » avec lien vers winelib



Paravirtualisation

Paravirtualisation (1)

- OS invité optimisé car il sait qu'il est virtualisé

Contient des APIs pour lier les VMs directement à l'hyperviseur

- Hyperviseur de **type 1** qui accède directement au hardware

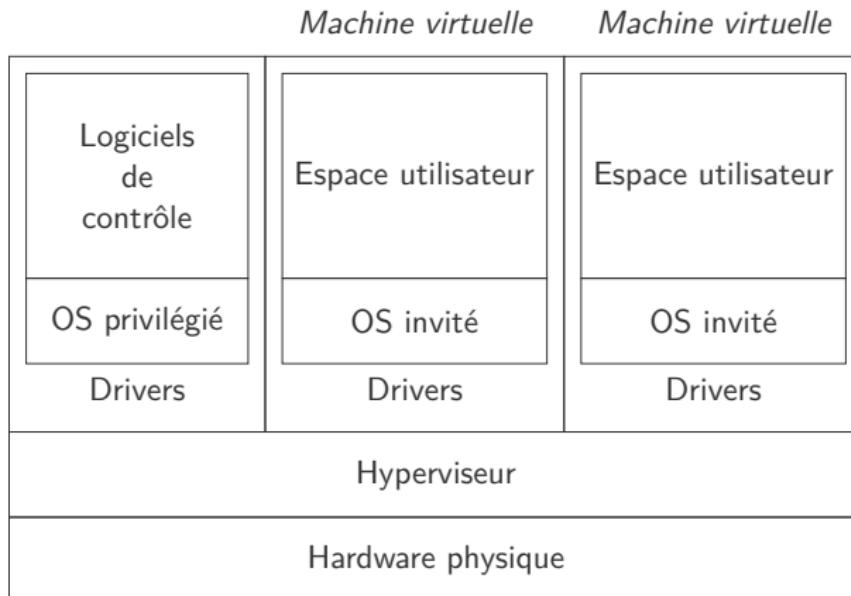
Bare-metal VMM

- Plusieurs **solutions existantes** éditées par des grosses firmes

Xen Server, VMWare Sphere, Microsoft Hyper-V Server

Parallels Server Bare Metal, KVM, Proxmox (KVM/LXC)...

Paravirtualisation (2)



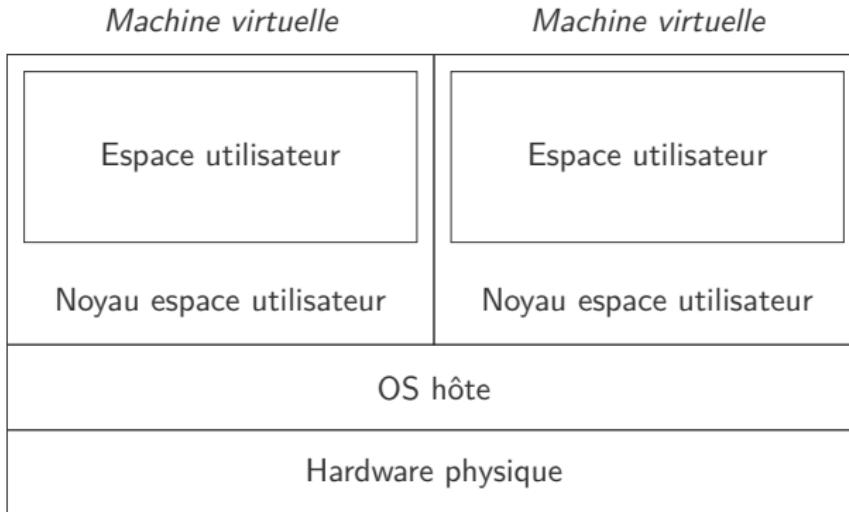
Paravirtualisation (3)

- Toutes les instructions conservées sont **virtualisables**
 - Il ne faut pas modifier les applications invitées
 - Plus rapide qu'exécution directe avec traduction binaire
 - Moins de changement de contextes
- L'**OS invité** doit être modifié pour communiquer avec le VMM
 - L'OS invité doit être porté sur le sous-ensemble x86 choisi
 - Il existe déjà pleins de Linux et BSD portés sur Xen !

Noyau dans l'espace utilisateur (1)

- Exécution d'un **kernel comme application** en mode utilisateur
 - Communication avec le kernel hôte*
- **User Mode Linux** est un exemple qui tourne sur Linux

Noyau dans l'espace utilisateur (2)



Conteneurisation



Virtualisation dans l'OS

- Exécution directe sur l'OS hôte avec **isolation**

Définition des ressources utilisables par les programmes invités

- **Limitation** des ressources utilisables et visibles

CPU, mémoire, réseau, disque, système de fichiers, pids...

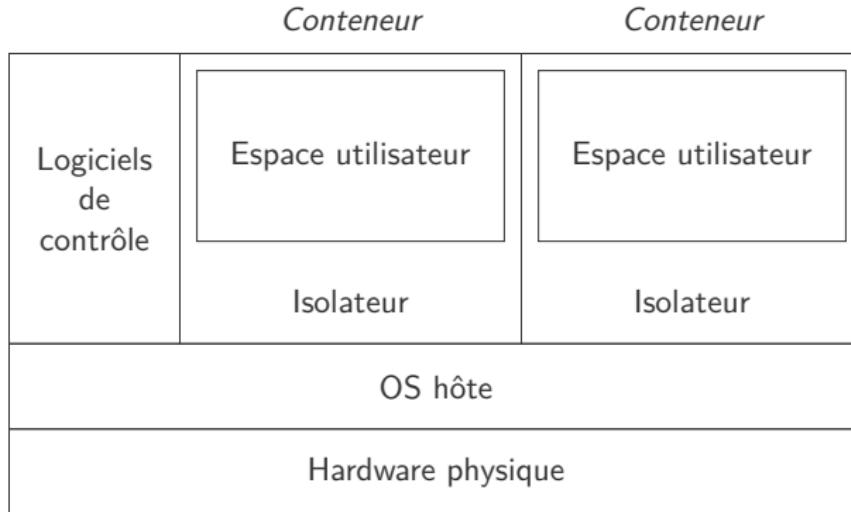
- Utilisation de **conteneurs** comme environnement isolé

Pas vraiment virtualisation car n'expose pas une machine

Conteneurisation (1)

- Un **conteneur** est un groupe de processus Linux
Ne sont pas des VMs puisque n'exécute pas un OS
- Plusieurs **mécanismes Linux** pour assurer l'isolation
 - **Groupe de contrôle (cgroups)** limite utilisation de ressources
Limite CPU, mémoire, réseau, disque, et donne priorités
 - **Espaces de nom** kernel limite visibilité de ressources
Ressources allouées à d'autres processus pas visibles
- Plusieurs logiciels de **conteneurisation** existants
LXC, CoreOS, LXC...

Conteneurisation (2)



Conteneurisation (3)

- Très **haute performance** pour l'exécution
 - Pas de changement de contexte entre app/OS invités et VMM
 - Pas démarrer un OS par app invitée, pas de ressources dédiées
- Les applications hôtes ne doivent **pas être réécrites**

Les snapshots sont petits, n'incluent pas l'état de l'OS
- Les app invitées doivent utiliser l'**interface de l'OS hôte**

LXC et Docker

- **Linux Containers** (LXC) est une technologie kernel

Exécution de pleins de processus dans des environnements isolés

- **Docker** est un outil pour packager une application

L'application et ses dépendances sont packagées ensemble

- Sur chaque machine physique, il n'y aura qu'**un seul kernel**

Technologie de virtualisation

- **Para**virtualisation

Modification légère du kernel invité pour optimisation

- Virtualisation **complète**

Aucune modification apportée au kernel invité

- Virtualisation par **conteneurs**

L'application et ses dépendances sont packagées ensemble

Crédits

- <https://www.flickr.com/photos/soldiersmediacenter/3966243098>
- <https://www.flickr.com/photos/catestorymoon/15679359872>
- <https://www.flickr.com/photos/catestorymoon/15228611595>
- <https://www.flickr.com/photos/82201175@N00/265388222>