TD R202 : Grands Principes de la virtualisation

Jean-Marc Pouchoulon

Mai 2023

Ce TD a été fortement inspiré par la conférence "Initiation à la virtualisation concrète" [1]

1 Virtualisation et processeur

Un processeur distingue quatre anneaux (ring 0 à 3) correspondant des niveaux d'exécution dont le plus privilégié est le ring 0 et le moins privilégié le ring 3.

Un programme est composé d'instructions en assembleur. Seules quelques instructions (par exemple les sorties vers un périphérique) nécessitent d'être exécutées en ring 0. Seul le Kernel Linux est autorisé à exécuter des instructions en ring 0. Chaque processus a donc une partie d'instructions s'exécutant dans l'espace Kernel pour l'accès au ring 0 et les autres dans l'espace utilisateur.

Un hyperviseur est un programme logiciel qui a pour but de faire fonctionner des machines virtuelles. Une machine virtuelle est assimilable à un processus pour l'hyperviseur. Par exemple la commande ps permet de visualiser ici le processus qui correspond à une machine windows 11:

```
ps -ef|grep kvm
pouchou    47352    2077 99 16:20 pts/0    01:42:40 /usr/bin/qemu-system-x86_64
                                          -name windows-11,process=windows-11 -pidfile windows-11/windows-11.pid
                                          -enable-kvm -machine q35,smm=on,vmport=off ...
```

1. Quelle est la solution de virtualisation utilisée ici ?
2. Que remarquez-vous ?

Ce processus veut exécuter parfois des instructions en ring 0 issues du kernel de la VM. Mais n'étant pas le kernel de l'hyperviseur il n'en a pas le droit et pourtant nous faisons de la virtualisation tous les jours... Une solution serait de tester chaque instructions devant être exécutées par la machine virtuelle afin de savoir si elle demande l'accès au ring 0 ou pas. L'idée est que l'hyperviseur émule un CPU, "trappe" les instructions privilégiées et les "émule".

1. Quel est l'inconvénient de cette méthode ?

Cette solution a prévalu jusqu'à ce que AMD et INTEL ajoute un jeu d'instructions dédié à la virtualisation. Ce jeu d'instructions dit VMX pour "Virtual Machine Extention" est présent sur vos machines si vous l'avez activé dans votre BIOS. L'idée est que quand le CPU rencontre une instruction privilégiée

1. https://www.youtube.com/watch?v=As44YzdnqqE

provenant de la VM il refuse de l'exécuter et il la transmet à l'hyperviseur. L'hyperviseur peut choisir d'exécuter l'instruction en émulant un CPU. Si l'instruction n'est pas privilégiée la VM l'exécute directement sur le CPU.

L'hyperviseur appelle l'instruction "VMX on" du processeur pour mettre en marche le mode virtualisation. On est passé alors en mode "VMX-ROOT" qui permet à l'hyperviseur d'accéder au RING 0. L'hyperviseur démarre la machine virtuelle en mode par l'instruction "VM Launch". Un des arguments de "VM Launch" est l'adresse de la première instruction à exécuter.

Le code de la VM s'exécute ensuite sur le CPU tant que les instructions ne sont pas privilégiées. On est maintenant dans le mode "VMX-NONROOT".

Si l'exécution d'une instruction privilégiée est demandé au CPU alors elle est transmise à l'hyperviseur qui effectue un "VM exit" et repasse en mode "VMX-ROOT". Il émule alors le CPU afin d'exécuter l'instruction privilégiée. Un "VM exit" est une opération coûteuse en termes de cycle CPU et il faut la limiter pour que la performance ne se dégrade pas trop. Un fois l'instruction émulée l'hyperviseur appelle l'instruction "VM resume" qui repasse en mode "VMX-NONROOT" et la lecture des instructions continue. Afin de préserver l'état des registres de la VM lors des "VM exit" et "VM resume" une structure : la VMCS (Virtual Machine Control Structure) permet de sauvegarder le contexte (registres..) de la VM et de l'hyperviseur.

1. Quel est l'intérêt du jeu d'instruction ?
2. Retrouvez dans votre bios l'endroit où est activé VMX.
3. Les jeux vidéos font beaucoup d'appels vers les périphériques (cartes graphiques, réseaux ..). Que pensez-vous de leur virtualisation ?
4. Retrouvez dans /proc/cpuinfo le jeu d'instructions VMX.

2 Virtualisation et mémoire

Chaque processus utilise des adresses mémoires. Cette mémoire a des adresses virtuelles [2] [3] et continues.

Dans le CPU c'est la "Memory Management Unit" qui assure la traduction de l'adresse virtuelle vers l'adresse physique. A cette fin la MMU interroge le "Translation Lookaside Buffer" qui est un cache de petite taille mais très rapide de la correspondance adresse virtuelle -> adresse physique. Si la recherche dans le TLB échoue la recherche continue dans une table en mémoire (Page Table).

Dans le cas d'une machine virtuelle les addresses virtuelles correspondent à des adresses physiques "bidons". C'est l'hyperviseur qui va se charger de gérer une table de correspondance adresses virtuelles - adresses physiques de l'hôte physique. Cette table est en lecture seule et quand la VM veut mettre à jour la "page table" une erreur est levée, le CPU interroge l'hyperviseur qui émet un "VM exit".

Ce mécanisme de "Shadow Page Table" est donc coûteux en termes de performances!

EPT/RVI est une réponse hardware à ce coût: les fondeurs [4] vont gérer deux tables dans la MMU:

— Une table de correspondance des adresses virtuelles de la VM en adresses physiques de la VM.

— Une table de correspondance des adresses physiques de la VM vers les adresses physiques du serveur.

La VM (le "guest") peut ainsi maintenir sa propre table des entrées sans faire appel à l'hyperviseur.

3 Virtualisation et devices

L'accès à un device physique entraîne un "VM exit" et une demande du CPU à l'hyperviseur. L'émulation logicielle d'un device (par exemple une carte réseau) est donc nécessaire. Mais développer et maintenir une carte réseau logicielle demande (trop) de temps [5].

2. il s'agit des adresses cela n'a rien à voir avec la virtualisation.
3. Chaque process peut disposer de 2^{64} adresses virtuelles
4. intel/amd
5. C'est l'intel pro E100 qui est émulée par l'hyperviseur mais c'est une carte très ancienne

Une solution c'est de "paravirtualiser": le kernel de la VM dispose d'un buffer qui permet l'échange d'informations entre la VM et l'hyperviseur.

Par exemple la VM échange des paquets réseaux dans un buffer partagé avec l'hyperviseur. L'échange se fait suivant une spécification (Virtio pour Linux, VMware Tools..).

Une autre solution est de permettre l'accès direct de la VM à un device (cartes réseaux, GPU...).

Un composant du CPU: l'IOMMU protège l'accès à la mémoire. Un device est réduit à accéder à une zone réservée pour la VM et ne peut pas accéder à la mémoire de l'hyperviseur.

L'inconvénient de cette méthode est que le device est "verrouillé" par la VM.

Une solution pour les cartes réseaux est l'utilisation d'outil comme "SR-IOV" qui permet d'instancier plusieurs cartes réseaux sur une seule carte p̂hysique. Chaque VM est donc connectée a une carte réseau virtuelle qui lui est dédiée.