Virtualisation

Professeur : Florent Gluck Assistant : Sebastien Chassot

17 mars 2022

Mini hyperviseur avec KVM

Introduction

Dans ce travail pratique, vous programmerez vous même, depuis presque zéro, un mini hyperviseur de type 2 (*hosted*) à l'aide de l'API KVM de Linux.

Le but de ce travail est que vous compreniez de manière plus approfondie les mécanismes de base de la virtualisation de plateforme, à savoir :

- le fonctionnement de l'API KVM et ses mécanismes
- la gestion d'un vCPU, notamment le passage entre hyperviseur et guest (`VMentry`), et le passage entre guest et hyperviseur (`VMexit`)
- l'émulation de périphérique
- la para-virtualisation de périphérique afin de simplifier le code et d'améliorer les performances
- le mécanisme d'hypercall
- les gains de performance et de simplicité de la para-virtualisation de périphérique par rapport à l'emulation

L'hyperviseur à implémenter est volontairement extrêmement rudimentaire pour des raisons de simplicité et de temps. Le but n'est donc pas d'émuler une plateforme matérielle existante complète comme un PC à l'image de ce que réalise QEMU, mais seulement quelques aspects de celui-ci.

En particulier, vous émulerez seulement certains aspects de deux périphériques d'un PC : l'affichage graphique et le contrôleur de disque. De plus, il s'agira seulement d'une émulation partielle et non complète/exhaustive, car vous verrez que de réaliser une émulation même partielle n'est pas trivial.

Cahier des charge

Le cahier des charges de ce travail pratique comprends les objectifs suivants :

1. Emulation partielle du mode texte VGA de 80 colonnes par 25 lignes en 16 couleurs :
   - L'OS guest doit pouvoir écrire des caractères à l'écran, à la position et à la couleur désirée et l'affichage obtenu doit être identique à celui obtenus sur un PC réel, que le guest écrive dans le framebuffer sur 8, 16, ou 32 bits
   - L'émulation doit correspondre au comportement attendu par le code présenté à la section "Mode texte VGA"
   - Le rafraîchissement de l'écran à 60Hz doit être émulé
   - Les aspects suivants n'ont pas à être émulés :
     - possibilité de changer la palette de couleurs
     - gestion du curseur
2. Emulation (très) partielle du contrôleur de disque ATA (IDE) :
   - L'écriture d'un secteur en mode PIO sur le premier disque doit être émulée

    - l'émulation doit correspondre au comportement attendu par le code présenté à la section "Ecriture d'un secteur"
    - on considère qu'un secteur possède une taille de 512 bytes
    - on considère que l'indexe du secteur à écrire est représenté sur 28-bits et le premier secteur se trouve à l'indexe 0
    - on considère qu'il n'y a qu'un seul disque
  - L'écriture de plusieurs secteurs consécutifs n'a pas à être émulée
  - La lecture d'un secteur n'a pas à être émulée
3. Implémentation paravirtualisée des deux périphériques ci-dessus : affichage et écriture d'un secteur
  - L'écriture sur le port 0xABBA (PMIO) par le guest permettra de notifier l'hyperviseur d'un hypercall
  - Le mécanisme et protocole pour transférer les arguments et la valeur de retour entre hyperviseur et guest est complètement libre
  - Pensez à écrire une fonction générique (p.ex. `hypercall`) permettant de réaliser des hypercalls
    - le but est de pouvoir facilement ajouter un hypercall sans avoir à changer l'implémentation de la fonction `hypercall`
    - si l'implémentation de la fonction doit changer lorsqu'un nouvel hypercall est ajouté, alors votre méthode n'est pas générique, donc incorrecte
4. L'hyperviseur, nommé `vmm`, doit gérer les arguments suivants, passés en ligne de commande :

```
vmm -guest BIN -disk FILE

BIN is a file that contains the guest OS to load and execute.

FILE is a file that's the content of the disk the VM must expose. It's simply a
     raw disk image and its size must be a multiple of 512 bytes (sector size).
```

5. L'exécution de la machine virtuelle doit pouvoir se terminer proprement via la pression sur une touche, p.ex. "ESC".
  - Cela doit typiquement être le cas lors des tests d'affichage VGA où le guest exécute une boucle infinie et ne se termine donc jamais.
6. Assurez-vous que votre hyperviseur passe tous les tests appelés dans le `Makefile` racine (cible `test_all`).
7. Réalisez des mesures de performances (benchmark), p.ex. écritures de 10'000 secteurs, afin de comparez les performances obtenues entre l'écriture de secteur émulée et paravirtualisée.
  - Ajoutez la ou les cibles permettant de réaliser ces benchmarks au `Makefile` racine afin que cela soit facilement testable.
  - Décrivez la méthodologie utilisée pour réaliser cette comparaison et mesurer les temps obtenus.
  - Pensez à compiler avec l'option `-O3` pour indiquer à `gcc` d'optimiser votre code.

Code et ressources à disposition

Afin de vous donner une base sur laquelle démarrer, le template ci-dessous vous est fourni sur le git du cours, dans le répertoire `labs/05-lab-hypervisor/template` :

```
|-- guest
|   |-- Makefile
|   |-- shared
|   |   |-- entrypoint_asm.s
|   |   |-- guest.ld
|   |   |-- ide_emul.c
|   |   |-- ide.h
|   |   |-- ide_pv.c
|   |   |-- pmio_asm.s
|   |   |-- pmio.h
```

```
|   |   |-- test_disk.c
|   |   |-- test_disk.h
|   |   |-- utils.c
|   |   |-- utils.h
|   |   |-- vga_emul.c
|   |   |-- vga.h
|   |   `-- vga_pv.c
|   |-- test_disk_emul.c
|   |-- test_disk_pv.c
|   |-- test_vga_emul.c
|   `-- test_vga_pv.c
|-- Makefile
|-- tests
|   `-- disk_ref.raw
`-- vmm
    |-- font.c
    |-- font.h
    |-- gfx.c
    |-- gfx.h
    |-- Makefile
    `-- vmm.c
```

Seuls les fichiers sources suivants sont autorisés à être modifiés :

```
guest/shared/vga_pv.c
guest/shared/ide_pv.c
guest/shared/utils.c
guest/shared/utils.h

vmm/vmm.c
vmm/Makefile

Makefile
```

Vous pouvez toutefois ajouter de nouveaux fichiers si besoin.

- Le répertoire `guest` contient différents OS guests permettant de valider l'émulation et la paravirtualisation de l'affichage VGA et du disque (écriture secteur) :
  - `entrypoint_asm.s` code assembleur du point d'entrée du noyau ; celui-ci initialise le MMU afin de mapper les adresses virtuelles sur les adresses physiques et d'appeler la fonction `guest_main` qui est le point d'entrée de l'OS guest en C (en réalité un noyau extrêmement primitif).
  - `guest.ld` fichier *linker* permettant de générer un exécutable "plat" (à contrario d'un fichier ELF) où on s'assure que le début du binaire commence par la section `entrypoint` définie dans `entrypoint_asm.s`.
  - `vga_emul.c` code d'affichage d'un caractère en mode texte VGA pour une carte VGA réelle.
  - `vga_pv.c` code d'affichage paravirtualisé d'un caractère en mode texte VGA ; à vous d'implémenter ce code.
  - `vga.h` constantes liées à l'affichage en mode texte VGA et prototypes des fonctions d'affichage de caractère.
  - `ide_emul.c` code d'écriture d'un secteur pour un contrôleur IDE réel.
  - `ide_pv.c` code d'écriture d'un secteur paravirtualisé.
  - `ide.h` taille d'un secteur et prototypes des fonctions d'écriture de secteur.
  - `test_disk.c` code réalisant les tests d'écriture de secteurs.
  - `test_disk.h` header pour `test_disk.c`
  - `pmio_asm.s` et `pmio.h` code assembleur, appelable depuis C, permettant d'écrire sur les ports.
  - `utils.c` et `utils.h` routines utilitaires, pour l'instant la fonction `memset` uniquement.
  - `test_vga_emul.c` OS guest qui réalise le test d'émulation de l'affichage VGA.
  - `test_vga_pv.c` OS guest qui réalise le test de paravirtualisation de l'affichage VGA.
  - `test_disk_emul.c` OS guest qui réalise le test d'émulation d'écritures de secteurs.

    - `test_disk_pv.c` OS guest qui réalise le test d'écritures de secteurs paravirtualisées.
- Le fichier `tests/disk_ref.raw` est le disque de référence pour les tests d'écriture de secteurs.
- Le répertoire `vmm` contient une ébauche d'hyperviseur :
    - `font.c` et `font.h` table de caractères ASCII étendus correspondant à la table 8-bit Code Page 437 de l'IBM PC d'origine. Chaque caractère est représenté par 16 valeurs de 8 bits où chaque valeur représente une ligne du caractère et chaque bit représente 1 pixel du caractère. Les 16 premières valeurs de la table représentent donc le 1er caractère, les 16 suivantes le deuxième, etc.
    - `gfx.c` et `gfx.h` *wrapper* au dessus de SDL2 permettant de créer une fenêtre et d'y afficher des pixels. Vous pouvez trouver un exemple d'utilisation, `gfx_example.c`, sur le projet gfxlib disponible sur github ici.
    - `vmm.c` contient l'ébauche de l'hyperviseur. La plupart du code qui permet de créer une machine virtuelle vous est déjà donné.

Nous vous fournissons également le code source d'un mini noyau bare-metal pour Intel IA-32 qui réalise exactement le même affichage en mode texte VGA que celui réalisé dans les tests de l'OS guest plus haut. La différence est qu'ici le noyau est exécuté par QEMU qui émule un vrai PC et vous pouvez ainsi voir à quoi devrait ressembler l'affichage. Tapez `make run` pour l'exécuter. Le code source du noyau se trouve dans `labs/05-lab-hypervisor/resources` :

```
resources/
`-- bare_metal_kernel
    |-- grub
    |   `-- grub.cfg
    |-- kernel
    |   |-- bootstrap_asm.s
    |   |-- kernel.c
    |   |-- kernel.ld
    |   |-- Makefile
    |   |-- types.h
    |   `-- vga.h
    `-- Makefile
```

Dans `resources`, vous pouvez également trouver :

- `VGA_text_mode.pdf` : slides décrivant la structure mémoire et la programmation du mode texte VGA.
- `VGA_16cols_pal.png` : image de la palette des 16 couleurs par défaut utilisées par le mode texte VGA. Vous pouvez utiliser le programme `gimp` pour inspecter les composantes RGB de chacune des couleurs.

Affichage en mode texte VGA

En mode émulation, votre hyperviseur doit émuler l'affichage des caractères correspondant aux valeurs écrites dans le framebuffer par le guest. L'affichage produit par votre hyperviseur devrait être identique à celui produit par QEMU pour le projet `bare_metal_kernel`.

En mode paravirtualisé, le code dans l'OS guest est beaucoup plus simple et lisible tout en étant plus efficace. La fonction d'affichage à implémenter vous est déjà donnée mais est vide, `putchar_pv`, dans `vga_pv.c`. Vous devez alors implémenter la partie *frontend* manquante dans le guest sous forme d'un hypercall et la partie correspondante (le *backend*) du côté hyperviseur.

Au niveau du rendu graphique de l'affichage, l'hyperviseur devrait gérer l'affichage dans un thread séparé et utiliser le code fourni dans `gfx.c` et `gfx.h`.

Ecriture d'un secteur

En mode émulation, votre hyperviseur doit d'émuler l'écriture d'un secteur (512 bytes) en mode PIO sur le premier disque IDE (ATA). Le code permettant d'écrire un secteur vous est déjà fourni dans le fichier `template/guest/shared/ide_emul.c`. Pour réaliser cette émulation côté hyperviseur, vous devrez implémenter une machine à état, similairement à ce qui a été illustré dans le cours.

Tout comme pour l'affichage, en mode paravirtualisé, le code dans l'OS guest doit être beaucoup plus simple et lisible que l'émulation, tout en étant beaucoup plus efficace. La fonction à implémenter vous est déjà donnée dans `ide_pv.c`. Similairement à ce qui est demandé pour l'affichage, à vous d'implémenter cette fonction pour réaliser l'hypercall correspondant.

Rendu

Sur le git de votre projet, vous rendrez à la **date indiquée sur la page Cyberlearn** du cours :

- Le code complet de votre travail
- Un rapport nommé `rapport.pdf` à la racine du git

Votre rapport doit être aussi **concis** que possible et comporter au minimum ces sections :

- **Etat** du projet
  - Est-ce que tous les points ont été réalisés ?
  - Est-ce que tout fonctionne comme demandé ? Si ce n'est pas le cas :
    - qu'est ce qui ne fonctionne pas correctement ou comme demandé ?
- **Conception**
  - Description des concepts pensés et mis en place pour la résolution des objectifs
- **Réalisation**
  - Pour les points importants et particulièrement pointus (ou difficiles à comprendre dans votre code), description de l'implémentation de ces différents points
- **Résultats**
  - Quelles mesures de temps ont été effectuées et quels temps ont été obtenus ?
  - Comparaison des résultats obtenus entre la version émulée et la version paravirtualisée de l'écriture de secteurs.
- **Conclusion**
  - Retour sur le projet
    - problèmes principaux rencontrés
    - points à améliorer

Important

Merci de ne **committer sur votre git que les fichiers sources**, donc pas de fichiers objets ou binaires ! (hormis le pdf ci-dessus).

Nous vous demandons d'être présent en cours afin de nous décrire, en live, l'état d'avancement de votre projet et votre code. Il est important que nous constations un **avancement continu** de votre travail de cours en cours.

Réalisez donc **régulièrement** des commits/pushes sur le git. Nous ne voulons pas voir un ou deux pushes effectués quelques minutes ou quelques heures avant le rendu final du projet ! Si vous êtes en groupe, les commits doivent être plus ou moins équilibrés entre les différents membres du groupe.

Enfin, le code rendu doit être le votre, c'est à dire propre à votre groupe. **La fraude est bien sûre interdite** et tous les codes rendus seront analysés par JPLAG.