Projet d'aprofondissement - 2011- HEPIA

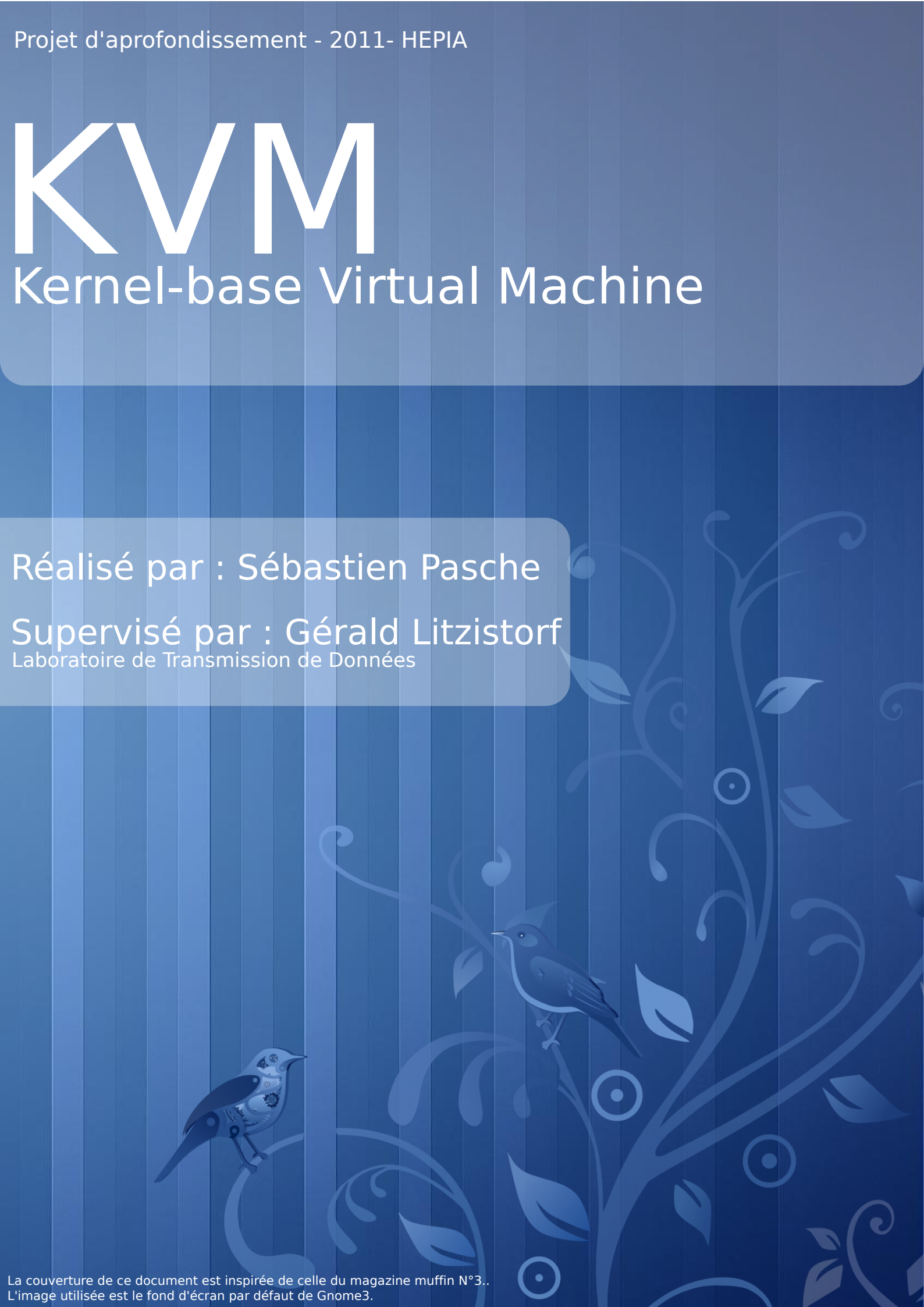
KVM

Kernel-base Virtual Machine

Réalisé par : Sébastien Pasche

Supervisé par : Gérald Litzistorf

Laboratoire de Transmission de Données



La couverture de ce document est inspirée de celle du magazine muffin N°3..
L'image utilisée est le fond d'écran par défaut de Gnome3.

h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

La virtualisation selon KVM (Kernel-base Virtual Machine)

Rapport technique

Réalisé par Sébastien Pasche et supervisé par le professeur Gérald Litzistorf

Version 1.0

Sébastien Pasche
Avenue Haldimand 39
1400 Yverdon-les-Bains
Switzerland

Executive summary

Ce travail d'approfondissement et de semestre avait pour principal but d'étudier et d'analyser la solution de virtualisation KVM de Red Hat. Cette analyse doit fournir un premier état de l'art des technologies et des décisions d'implémentations sur lesquels KVM a été construit. KVM est aujourd'hui une solution de virtualisation soutenue et utilisée par de grands acteurs du marché informatique (IBM, Intel, HP, etc.), il est donc intéressant de s'y intéresser de prêt. Cette analyse couvrira les éléments de bases nécessaires pour construire un environnement de virtualisation animé par KVM. Chacun de ces éléments sera étudié du point de vue de son architecture (technologie, principes, paradigmes, etc.) et de son implémentation (code source, API proposée, etc.)

Ce travail à passer en revue les briques de bases d'un environnement de virtualisation fondé sur KVM du point de vue de leurs architectures et de leurs implémentations. Cette étude a permis de relever les forces, les faiblesses ainsi que les évolutions futures prévues et possibles de cette solution de virtualisation.

Les résultats de ces analyses ont montré une solution de virtualisation moderne, construite de manière pragmatique et ayant résolu des problématiques d'ampleur comme les mises à jour du moteur de virtualisation ou les fortes montées en charges. Pour résoudre ces problèmes KVM a été construit comme une composante interne de Linux disposant ainsi de tout le savoir faire acquis et futur de ce système.

La seconde partie du travail a permis de spécifier et de lancer le chantier de la migration des laboratoires de virtualisation de l'HEPIA vers KVM. Cette partie décrit aussi les mécanismes nécessaires pour automatiser l'installation de l'environnement de virtualisation KVM (serveurs et workstation) ainsi que la création des machines virtuelles à l'aide d'outils open-source soutenus par les grands acteurs actuel de l'informatique.

Ce document permet donc de découvrir comment est construit KVM et à quoi peut ressembler un environnement KVM de grande et moyenne taille.

Introduction

It is change, continuing change, inevitable change, that is the dominant factor in society today. No sensible decision can be made any longer without taking into account not only the world as it is, but the world as it will be.

Isaac Asimov.

Il était une fois...

C'est ainsi que dans l'inconscient collectif commence la plupart des histoires et c'est probablement la meilleure façon pour commencer ce travail. La thématique principale de ce document est la virtualisation et, la virtualisation, bien qu'étant un sujet d'actualité, est une vieille histoire comparativement à l'histoire de l'informatique moderne. Imaginez : déjà dans les années 70-80 avec les CP/CMS [1] de chez IBM [2] commençaient à paraître les premières solutions de virtualisation commerciales. Dans le chapitre Histoire de la virtualisation, nous verrons que les premières traces de la virtualisation remonte à plus de 50 ans, ce qui, dans l'informatique, représente "une éternité". Il serait même donc possible de changer le titre du chapitre par : "Il était une fois, il y a fort longtemps ...".

Aujourd'hui, des solutions de virtualisation clés en main sont à disposition des utilisateurs privés et des administrateurs réseaux. Mais en réalité, ces nouvelles solutions côtoient encore dans les environnements de production des solutions plus anciennes comme celles utilisées sur les gros mainframes IBM [3].

Commençons par définir la virtualisation puisque c'est le sujet de ce travail.

Virtualisation
Parmi les nombreuses définitions de la virtualisation disponibles sur internet et dans la littérature, il a été choisi d'adopter pour ce travail celle qui paraissait la plus réaliste et adaptée. Cette définition est celle proposée par Amit Singh [4], travaillant actuellement dans le département "Operating systems developpement" de Google. Amit Singh a aussi travaillé au "IBM Almaden Research Center" et au "Information Sciences Research Center of Bell Laboratories" et est reconnu comme un spécialiste du monde de la virtualisation sous Linux.

1. Article Wikipédia à propos des IBM 360 et 370 : http://fr.wikipedia.org/wiki/IBM_360_et_370
2. Site officiel de IBM : http://www.ibm.com/
3. Numéro special de Market sur le monde bancaire : http://www.market.ch/fileadmin/documents/market.ch/pdf/MAF80_pIBCOM.pdf
4. Blog personnel de Amit Singh : http://www.kernelthread.com/

> *« Amit Singh 2004 [a] Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others. »*
>
> *a.* http://www.kernelthread.com/publications/virtualization/

Il est facile de remarquer que cette définition est très globale et englobe beaucoup de sujets connexes. Prenons, par exemple, la "qualité de service" qui pourrait être en elle-même un sujet d'étude à part. Cependant, un point intéressant est que la qualité de service est aujourd'hui souvent mise en place à l'aide d'infrastructures virtualisées [5]. En d'autres mots, le concept de virtualisation est aujourd'hui liée à de nombreuses problématiques à tel point qu'il devient pratiquement inévitable dans l'informatique.

Un point important dont il faut tenir compte à propos de cette définition est qu'elle n'est pas parfaite. Ce domaine est large et varié dans lequel beaucoup de notions se regroupent sous la large définition de la virtualisation.

Par exemple, prenons le cas de la distributivité proposée par Amit Singh dans sa définition. La virtualisation n'implique pas forcément la notion "partitionnement" ou de "répartition" et dans certains cas, peut même impliquer l'inverse. Un cas imaginable est celui qui consiste à prendre plusieurs disques durs et les faire apparaître comme un seul sous une couche/interface virtuelle (donc distribuer la charge de travail sur plusieurs disques) appelée RAID. Interprété comme cela, le RAID est une forme de "virtualisation" qui entrerait dans cette définition très générale.

Pour des questions de limitations dans le temps, il a été choisi de prendre certains raccourcis ou d'élimer des notions théoriques et pratiques de ce document. Le contenu de ce rapport ne sera donc pas un cours sur la virtualisation ou une documentation exhaustive de KVM.

Aujourd'hui, il n'est plus possible d'ouvrir un magazine IT ou une newletter d'une grande entreprise informatique sans entendre parler de virtualisation. Le chapitre suivant tente de regrouper un certain nombre d'arguments en faveur de la virtualisation qui ont été condensés et traduits à partir de multiples lectures sur internet, de " Whitepapers" et de sites web de grandes enseignes informatiques.

Pourquoi choisir la virtualisation

Aujourd'hui, on retrouve énormément de justifications à la virtualisation, qu'elles soient uniquement mercantiles ou techniquement justifiées. Afin d'entrer dans la lecture de ce document avec l'envie et la motivation d'utiliser la virtualisation, il peut être intéressant de rappeler une partie de ces justifications :

- Les machines virtuelles peuvent être utilisées pour consolider un système subissant une montée en charge ou se prémunir contre la montée en charge. Ceci apporte bien sûr un bénéfice "relatif" (qu'il soit perçu ou réel comme beaucoup de vendeurs le disent) en termes de coûts hardware, environnementaux [6] ainsi que de management et de la gestion des infrastructures concernées.

- Le besoin d'être capable de faire fonctionner de vieilles applications ne pouvant plus être utilisées sur des systèmes modernes. Certaines vielles applications ne peuvent en effet plus fonctionner sur le matériel, ou les systèmes d'exploitation actuels. La virtualisation permet de réutiliser ces applications sans forcément avoir à racheter du matériel ou sous-utiliser du matériel compatible s'il en existe. Certains vieux systèmes ne sont pas capables de fonctionner à plusieurs sur une même machine (par exemple, des applications avec des IPC Systèmes codés en dur).

5. Exemple de publication (2003) très général traitant des couches réseaux, de la virtualisation et de la qualité de service liée : http://www.ecsl.cs.sunysb.edu/tr/TR162.pdf
6. Rapport du congres "Energy Star" de 2007 : http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf

IPC : En informatique, les communications inter processus (Inter-Process Communication ou IPC) regroupent un ensemble de mécanismes permettant à des processus concurrents (ou distants) de communiquer. [7]

- Les machines virtuelles peuvent être utilisées pour apporter de la sécurité (Isolation, partitionnement) autour d'applications considérées comme peu sûres ou peu stables. Dans certains cas, il est même possible de créer ce genre d'environnement à la volée. Par exemple, vous téléchargez un logiciel sur internet et il serait automatiquement isolé dans un système virtuel. De ce point de vue, il est possible d'imaginer de nombreuses solutions, comme l'obfuscation d'adresses mémoire par exemple. La virtualisation est aujourd'hui un excellent vecteur pour la création d'environnements d'exécution sécurisés.
- La virtualisation peut être utilisée pour créer des systèmes ou des environnements d'exécution disposant de ressources limitées. De plus, ces ressources peuvent être dimensionnées dynamiquement en fonction des besoins. Le partitionnement des ressources permet d'aller dans la même direction que ce que vise la qualité de service.
- La virtualisation peut être utilisée pour donner à des applications l'illusion d'un hardware spécifique (par exemple, des device SCSI, plusieurs processeurs, etc.). La virtualisation peut aussi être utilisée pour simuler des réseaux complexes composés de multiples nœuds et services. Ces cas d'utilisation sont utiles pour des environnements de tests ou des environnement de productions complexes.
- Les machines virtuelles donnent la possibilité de faire fonctionner différents systèmes d'exploitations en même temps, par exemple : de différentes versions ou des systèmes complètement différents. Certains d'entre-eux peuvent, entre autres, être des solutions de backup à chaud tandis que d'autres pourraient être simplement impossibles à installer sur du matériel neuf.
- La virtualisation permet de "debuger" ou de superviser les performances et l'état d'un système en cours de fonctionnement. Les outils actuellement disponibles permettent de faire du "debug" et du "monitoring" avancé sur des systèmes virtualisés sans pour autant avoir des impacts négatifs sur la productivité du système en production.
- Les machines virtuelles fournissent une isolation aux applications qu'elles animent. De cette manière, les erreurs et les crashs sont contenus. Il est aussi possible de créer des clones dans lesquels des fautes seraient injectées pour en observer les effets.
- La virtualisation et les machines virtuelles rendent plus facile et fluidifient les procédures de migration. De cette manière, la virtualisation rend les applications et les systèmes plus mobiles en général. Avec les outils actuellement disponibles, il est possible de traiter les applications comme des "appliances" qui seront déployées chacune dans un conteneur virtuel dédié.
- De manière générale, les machines virtuelles sont des outils très puissants pour le développement, la recherche et les milieux académiques. De plus, l'isolation qu'elles proposent et leur capacité à être facilement manipulées permettent de faire facilement des sauvegardes d'un état précis d'un système.
- Actuellement, la virtualisation permet de faire fonctionner tous les systèmes d'exploitation courants sur des ressources partagées.
- Les machines virtuelles peuvent être utilisées pour créer des scénarios de tests/désastres/évolution/validation afin d'améliorer la qualité globale des services proposés par une infrastructure et de proposer des solutions de continuité.
- La virtualisation permet de manière simple d'ajouter de nouveaux services à un catalogue existant sans demander un surplus de travail au niveau systèmes (clonage + déploiement du nouveau service).
- Les solutions actuelles de sauvegarde et de gestion des machines virtuelles permettent de faire simplement et rapidement des "backups" ou restauration. Ce point rend les tâches de migration plus simples et moins risquées .
- La virtualisation a permis la démocratisation de la location de serveur distant (par exemple, le cloud amazon) et la notion de SAAS(Software as a service).

7. Artcile Wikipedia à propos des IPC : http://en.wikipedia.org/wiki/Inter-process_communication

- La virtualisation est une technologie actuellement à la mode.
- Beaucoup d'autres raisons.

Les chapitres suivants de cette introduction présentent les différents objectifs que doit couvrir ce travail de semestre et la forme qu'il prendra, partie par partie.

Objectifs du travail

Ce travail de semestre a pour but d'étudier et de valider une couche d'infrastructure de virtualisation open-source qui sera utilisée pour de futurs projets. `HEPIA` est en effet intéressée à remplacer ses laboratoires actuellement sous `VMware` par une solution open-source viable. En plus de ces besoins d'infrastructure, le laboratoire de virtualisation de l'HEPIA mène actuellement des projets liés à la virtualisation où le besoin de manipuler à sa guise l'outil de virtualisation devient nécessaire.

Travailler avec des solutions open-source apporte, outre le coût direct, d'autres avantages, comme par exemple :

1. Comprendre comment les mécanismes de virtualisation sont mis en œuvre.
2. La contribution constante du monde open-source permet une évolutivité du système dans le temps et un suivi des nouvelles technologies.
3. Le code source est disponible et donc est auditable à tout moment.
4. etc.

Valider une solution de virtualisation demande de comprendre comment elle est implémentée, mais aussi d'évaluer sa possible durée de vie ainsi que sa capacité à être utilisée en production à grande échelle.

Ce travail de semestre (projet d'approfondissement) traite de la solution de virtualisation KVM (Kernel-base virtual machine) [8] comme couche d'infrastructure de virtualisation. KVM est une solution de virtualisation novatrice utilisant une approche moderne de la virtualisation bien que reposant sur des principes historiquement fondateurs. Ce document a pour but de montrer si `KVM` est une solution de virtualisation viable, sécurisable, maintenable et utilisable à grande échelle tout comme le serait une solution propriétaire comme par exemple VMware ESX [9].

Aujourd'hui, la plupart des solutions de virtualisation de grande taille reposent sur une architecture équipée d'un SAN. Dans le cadre de ce projet, il a été choisi, dans le cas où le temps le permettrait, d'étudier une solution novatrice dans le monde des architectures virutalisées. Il s'agit de sheepdog [10] une solution de stockage distribuée qui pourrait être comparée à un cloud [11].

`Architectures distribuées` : L'Architecture d'un environnement informatique ou d'un réseau est dite distribuée quand toutes les ressources ne se trouvent pas au même endroit ou sur la même machine. On parle également d'informatique distribuée. Ce concept s'oppose à celui d'architecture centralisée dont une version est l'architecture client-serveur. [12]

`SAN` : En informatique, un réseau de stockage, ou SAN (de l'anglais Storage Area Network), est un réseau spécialisé permettant de mutualiser des ressources de stockage. [13]

8. Site officiel de KVM : https://www.linux-kvm.org/page/Main_Page
9. Site officiel de `VMware` : http://www.vmware.com/
10. Site officiel de SheepDog : http://www.osrg.net/sheepdog/
11. Artcile Wikipedia à propos du cloud computing : http://en.wikipedia.org/wiki/Cloud_computing
12. Artcile Wikipedia à propos des architectures distribuées : http://fr.wikipedia.org/wiki/Architecture_distribuée
13. Article Wikipédia à propos du SAN : http://en.wikipedia.org/wiki/Storage_area_network

Ce travail de semestre est donc un travail de veille [14] technologique qui vise à aider le laboratoire de virtualisation de l'HEPIA et les personnes intéressées à acquérir une base de connaissances nécessaire à comprendre et utiliser KVM comme solution de virtualisation. Ce socle de connaissances doit permettre de commencer une étude et un apprentissage avancé de l'utilisation de KVM, tout en étant capable de comprendre et maîtriser ses limites et ses forces.

Ce document donne un socle et suffisamment de références pour permettre au lecteur intéressé d'aller plus loin et de maîtriser l'outil de manière large et globale. Comme tout ne pourra pas être expliqué dans ce document, il a été choisi qu'en plus des références disponibles sur internet, le travail se basera sur un choix de livres et de documents. Le prochain chapitre explique quels sont les ouvrages sélectionnés ainsi que leurs références.

Sélection de livres utiles au sujet de ce travail

Utiliser, comprendre et administrer un système de virutalisation basé sur KVM demande des bases avancées du monde Linux. Le temps disponible pour ce projet de semestre ne permet clairement pas de couvrir tous ces points et de revenir sur des bases nécessaires à la compréhension de certains détails particuliers. Lorsqu'il n'est pas possible d'entrer dans les détails d'un point, soit un lien internet soit une référence littéraire sera donné.

Pour permettre de pointer sur des ressources très précises, il a été choisi de sélectionner un certain nombre d'ouvrages références. Ces livres sont disponibles dans le laboratoire de virtualisation de HEPIA, auprès de l'auteur de ce document et dans les bibliothèques universitaires avec un rayon informatique suffisamment bien rempli.

Administration et base de Linux
Administrer un ensemble de serveurs Linux ou même un seul serveur Linux, demande de maîtriser certains outils ou certains principes fondamentaux, tels que par exemple :

- Le partitionnement sous Linux.
- La gestion du stockage sous LVM.
- Le démarrage de Linux.
- La sécurité sous Linux, que cela soit du point de vue des droits ou celui de SELinux.
- etc.

Comme il n'est pas possible de tout maitriser mais qu'il est nécessaire d'avoir un bon point de vue généraliste pour retrouver la bonne documentation au bon moment, un choix de 4 livres a été réalisé sur la thématique de l'administration de systèmes Linux. Ces différents livres couvrent de manière large et étendue le sujet de l'administration de système Linux. La pédagogie dont fait preuve l'auteur ainsi que la structure choisie pour structurer les documents méritent d'être relevés pour leurs grandes qualités.

- Administration Linux Tome 1 - ISBN : 978 - 2212126242.
- Administration Linux Tome 2 - ISBN : 978 - 2212128826.
- Administration Linux Tome 3 - ISBN : 978 - 2212122459.
- Administration Linux Tome 4 - ISBN : 978 - 2212122480.

14. Article Wikipedia à propos de la veille technologique : http://fr.wikipedia.org/wiki/Veille_technologique

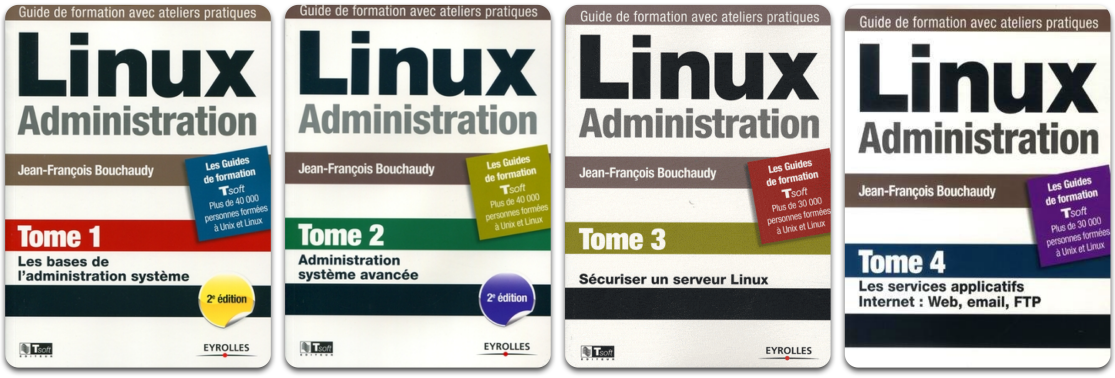


FIGURE 1: Les 4 tomes de Administration Linux aux éditions Eyrolles

Le fonctionnement du noyau

KVM repose en grande partie sur la structure existante du noyau Linux. Pour être capable de bien réagir en cas de problèmes ou de bien comprendre les implications des choix qui sont pris, il est nécessaire de connaître comment fonctionne le noyau de Linux. Pour ce faire, un seul livre qui couvre avec précision le domaine du noyau Linux a été sélectionné.

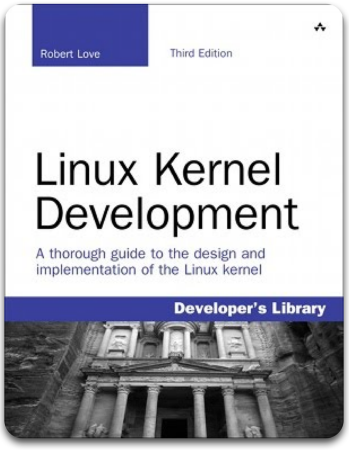


FIGURE 2: Linux Kernel Development ISBN-13: 978-0672329463

Interaction entre le noyau Linux et le système
Une fois les bases de l'administration Linux comprises, l'utilisateur familiarisé avec le fonctionnement général du noyau, il est encore nécessaire de franchir une étape supplémentaire pour être capable de comprendre du code source se trouvant dans le noyau de Linux ou travaillant en interaction étroite avec celui-ci. Le dernier ouvrage de la sélection proposée comme complément de documentation à ce rapport couvre effectivement le développement et les interfaces offertes aux développeurs pour interagir dans et avec le noyau Linux.

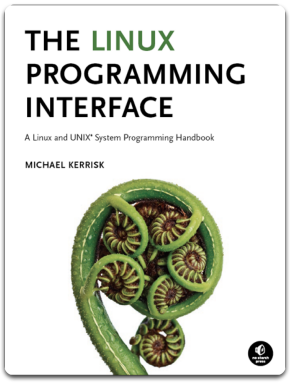


FIGURE 3: The Linux Programming Interface: A Linux and Unix System Programming Handbook - ISBN-13: 978-1593272203

Composition du document et du rapport

Le présent rapport est découpé en deux grandes parties : le rapport en lui même et, un résumé exécutif.

Le rapport technique
Le rapport technique comporte un aperçu global des études, des analyses et tests effectués lors du travail de recherche. Ce travail de semestre étant construit comme une veille technologique, le rapport est donc orienté dans une démarche d'analyse.

Résumé exécutif
Le résumé exécutif est une version vulgarisée du travail effectué lors de ce travail de recherche. Son but est de présenter le domaine traité de manière suffisamment simple et synthétique pour que des gens du domaine informatique avec de l'expérience "utilisateur" dans le domaine de la virtualisation puissent comprendre les tenants et aboutissants du projet.

Partie I - État de l'art et historique de la virtualisation

Sans vouloir être un cours sur la virtualisation, cette partie du document traitera globalement de l'histoire de la virtualisation et des solutions existantes en termes de technologies de virtualisation.

Ce chapitre essayera aussi d'expliquer les grands principes de la virtualisation afin de permettre au lecteur de comprendre globalement le contenu du document. Il est en effet difficile d'analyser et d'étudier des solutions de virtualisation complexes sans en connaître les principes fondamentaux.

Ce travail visant un but de "veille technologique", il est donc nécessaire de poser certaines bases avant de pouvoir entrer dans un sujet complexe tel que l'architecture de KVM ou de Sheepdog.

Partie II - Architecture de KVM

Cette section du document se repose sur les bases expliquées dans l'état de l'art précédent et sur les connaissances du lecteur pour construire une approche détaillée de l'architecture de KVM. L'environnement KVM étant à la fois simple (dans la manière dont il est construit) et complexe (pour les techniques et les implémentations qu'il utilise), son étude est un chapitre avec une forte orientation technique. Cette partie du document veut aussi donner certaines clés pour découvrir des outils open-source indispensables à l'utilisation correcte de KVM et aux bases de l'administration d'un système de virtualisation basé sur KVM.

Partie III - Utilisation de KVM

La troisième partie du document vise à mettre à l'épreuve la pratique KVM. Il ne s'agit pas ici de créer un guide complet (beaucoup de guides très efficaces existent sur Internet), mais de montrer par l'intermédiaire de cas simples la pratique de l'utilisation de KVM au quotidien. Cette section introduira au fur et à mesure des cas traités des outils permettant l'utilisation et l'administration d'un système KVM complet.

Guide de lecture et symbolique utilisée

Dans ce rapport, les normes graphiques suivantes sont utilisées dans le texte :

1. Les noms de logiciel sont mis en valeur de la manière suivante : KVM et Sheepdog.
2. Si des méthodes, des mots-clés ou des phrases doivent être mis en évidence, la forme suivante est utilisée : `Ceci est un test de mise en évidence` .
3. Les renvois en bas de pages sont décrits comme suit [15].
4. Mises en valeur diverses : Ceci est une mise en valeur.

Les citations sont présentées selon la forme suivante :

> *«Lady Mary Wortley Montagu :*
> *General notions are generally wrong.*
> *– letter, March 1710»*

En cas de besoin, des boîtes informationnelles peuvent être présentes sur les pages.

Ceci est un complément d'information dont la lecture n'est pas forcément indispensable. On aura recours au pictogramme ci-contre pour attirer l'attention du lecteur distrait...

Dans le cas où des commandes Unix/Linux sont présentées, elles sont présentées de la manière suivante :

```
grep -wi bidule /tmp/truc.dat | sort -n
```

Dans le cas où des fichiers de configuration/codes doivent être présentés, ils sont présentés de la manière suivante :

```
serviceName=win32cifs
hostname=0.0.0.0    # IP address of your AD server
username=user       # User with read access on AD Eventlogs
password=pwd        # Password of the provided user
domain=dom          # Windows domain
facility=Security   # Eventlog file to poll
```

15. Ceci est une note de base de page

Les versions utilisées

Voici les versions des différents logiciels utilisés et/ou analysés dans ce travail.

- Système : Fedora 14
- Code Kernel : linux-2.6.38.4
- Code qemu-kvm : qemu-kvm-0.14.0
- Code libvirt : 0.8.3

Sommaire

I

État de l'art
Virtualisation, du prestidigitateur au mime

Chapitre

1

Sommaire

Petite histoire de la virtualisation

All experience is an arch wherethrough gleams that untravelled world whose margin fades for ever and for ever when I move.

Alfred Lord Tennyson.

La virtualisation est une technologie qui aujourd'hui entre dans les habitudes des professionnels du domaine informatique. Cependant, d'un point de vue historique, elle est aussi passablement ancienne. Ce chapitre essaie de présenter un bref historique de la virtualisation.

Bien qu'ayant été globalement toujours un domaine d'effervescence, la virtualisation a eu des périodes plus agitées que d'autres. Actuellement, en 2011, nous entrons dans une de ces périodes par l'émergence de nouveaux leaders et de nouveaux marchés. Cependant, avant de commencer l'histoire de la virtualisation, il a été choisi de revenir sur la précédente vague d'activités qui avait alors eu lieu autour de 2003.

En 2003 `Microsoft` acquiert `Connectix Corporation`[1], un fournisseur de logiciels de virtualisation pour Windows et Macintosh. A la même époque, `EMC`[2] annonçait son plan de rachat de `VMware` pour 635 millions de dollars[3]. Peu de temps après, `VERITAS` (qui depuis a été racheté par `Symantec`) achetait `Ejascent` pour 59 millions de dollars[4]. Au cours de la même période, `Hewlett-Packard`[5] et `SUN`[6] (A présent `Oracle`[7]) travaillaient pour améliorer leurs propres solutions de virtualisation.

Aujourd'hui, en 2011, la virtualisation est toujours un sujet d'actualité et pratiquement toutes les grandes sociétés informatiques essaient de se placer sur ce marché.[8][9][10][11][12][13][14][15][16]

Il est donc intéressant, pour comprendre la virtualisation d'aujourd'hui, de revenir brièvement sur son histoire.

1. Article Wikipédia à propos de `Connectix Corporation` : http://en.wikipedia.org/wiki/Connectix
2. Article Wikipédia à propos de `EMC` : http://en.wikipedia.org/wiki/EMC_Corporation
3. Rachat de `VMware` par `EMC` : http://www.vmware.com/company/news/releases/emc.html
4. Rachat de `Ejascent` par `VERITAS` : http://www.thewhir.com/web-hosting-news/ver010904
5. Article Wikipédia à propos de `HP` : http://en.wikipedia.org/wiki/Hewlett-Packard
6. Article Wikipédia à propos de `SUN` : http://en.wikipedia.org/wiki/Sun_Microsystems
7. Acquisition de `SUN` par `Oracle` : http://en.wikipedia.org/wiki/Sun_acquisition_by_Oracle
8. http://www.bitpipe.com/tlist/Virtualization.html
9. http://www.bitpipe.com/tlist/Virtualization-Platforms.html
10. http://www.bitpipe.com/tlist/Virtualization-Security.html
11. http://www.bitpipe.com/tlist/Virtualization-Software.html
12. http://searchservervirtualization.bitpipe.com/
13. http://www.bitpipe.com/tlist/Server-Virtualization.html
14. http://www.bitpipe.com/tlist/Virtual-Machine.html
15. http://www.bitpipe.com/tlist/VM.html
16. http://www.bitpipe.com/tlist/Server-Virtualization-Software.html

Ce présent chapitre se charge de retracer les grandes étapes de la virtualisation et quelques problématiques importantes qui ont motivé sa continuelle innovation. En aucun cas ce document ne se veut exhaustif. Le but recherché ici est de montrer que bien que des innovations soient constamment apportées, les fondements utilisés par la virtualisation sont les mêmes qu'il y a plus de 50 ans.

Les technologies, ainsi que les entreprises mises en évidence dans cet historique, ont été choisies par l'auteur. Ces choix ont été principalement guidés par la thématique de ce travail afin de maintenir une vision et un point de vue focalisé sur KVM mais traitant autant du monde académique que professionnel.

1.1 La virtualisation, une problématique globale et historique

Malgré toutes les bonnes raisons qu'a la virtualisation d'exister, elle comporte en elle-même des problématiques fondamentales que les chercheurs et les administrateurs d'infrastructures informatiques essaient de résoudre depuis leurs balbutiements (Voir Chapitre 1 : Historique de la virtualisation). Ce paragraphe essaie de les présenter de manière résumée pour que le lecteur les garde en tête lors de la lecture du document et lors de l'utilisation de solution de virtualisation en général.

En 1974, Gerald J. Popek [17] et Robert P. Goldberg sortirent une série de publications, dont une nommée "Survey of Virtual Machines Research" [18], publiée dans le magazine COMPUTER. Cette publication décrit de manière pragmatique la notion de virtualisation, son utilisation et les critères minimaux à remplir pour utiliser la virtualisation. Leurs travaux sont aussi connus comme les "virtualization requierments" [19], soit les critères minimaux de virtualisation.

Il est intéressant d'étudier le passage suivant de la publication de Robert P. Goldberg :

> *« Robert P. Goldberg [a] ...Virtual machine systems were originally developed to correct some of the shortcomings of the typical third generation architectures and multi-programming operating systems.. »*
>
> a. https://www.cs.duke.edu/courses/spring11/cps210/papers/Gol74.pdf

Cette citation présente d'une manière très courte les besoins (voir la liste de la section précédente) qui mènent les gens à aller encore aujourd'hui vers des solutions de virtualisation. Un point amusant dans son approche est que l'un des arguments les plus importants aux yeux de Goldberg est le besoin de machines virtuelles pour répondre aux futurs besoins de répartition de charge et de parallélisation.

A partir des différentes publications de Gerald J. Popek et Robert P. Goldberg, il est possible de ressortir trois principaux critères à une virtualisation réussie :

- Le premier est ce qu'ils appellent le critère d'équivalence. Ceci implique qu'une application quelconque doit s'exécuter de la même manière, qu'elle soit exécutée au-dessus d'un hyperviseur ou d'une machine physique.
- Le second est celui du «contrôle de ressources». Ce critère indique que l'hyperviseur doit avoir le contrôle exclusif des ressources à partager. N'importe quelle application doit donc passer par cet outil pour pouvoir accéder à une ressource partagée.
- Le troisième et dernier principe est le critère d'«efficacité». Il indique qu'une part majoritaire d'instructions doit être exécutée par le processeur sans intervention de l'hyperviseur. Ce critère exclut les techniques de virtualisation totale. La paravirtualisation, elle, permet de répondre un peu plus à ce principe.

Robert P. Goldberg souligne dans son document que les systèmes permettant de faire fonctionner des logiciels de virtualisation doivent proposer deux modes de fonctionnement hardware: un mode privilégié et un mode non privilégié. Cette notion ramène directement à la notion de "Ring" des processeurs modernes. Aujourd'hui, tous les processeurs modernes proposent ces deux états.

`Ring des processeurs` : Un anneau de protection (ou ring) est l'un des niveaux de privilèges imposé par l'architecture d'un processeur. L'objectif est ici de ne donner accès qu'à une quantitié limitée d'instructions proposées par un processeur. [20]

17. Article Wikipédia à propos de Gerald J. Poepk : http://en.wikipedia.org/wiki/Gerald_J._Popek
18. Le numéro 6 de COMPUTER contenant cet article en page 34 est lisible à l'adresse suivantes : https://www.cs.duke.edu/courses/spring11/cps210/papers/Gol74.pdf
19. Article wikiédia consacré aux "virtualization requierments" : http://en.wikipedia.org/wiki/Popek_and_Goldberg_virtualization_requirements
20. Article Wikipédia à propo des RING : http://en.wikipedia.org/wiki/Ring_(computer_security)

Les concepts expliqués dans le document de Robert P. Goldberg permettent aussi de ressortir les recommandations et constations présentées dans le paragraphe suivant.

Pour montrer la clairvoyance et l'utilité des conseils de l'époque, ils sont directement mis en relation avec des problématiques actuelles de la virtualisation. L'extraction et la traduction de ces éléments est subjective et est le travail du rédacteur de ce document. La représentation de la forme n'est donc pas garantie à 100%.

- A) 1974 : Uniquement une seule interface au matériel ("bare machine interface") est exposée. De ce fait, un seul système peut fonctionner directement sur le matériel. De ce même fait, un programme ne peut fonctionner qu'en dessus du système.
- A) 2011 : Il n'est aujourd'hui toujours pas possible de se passer d'un hyperviseur, qu'il soit matériel ou logiciel, pour réussir à faire fonctionner deux systèmes en même temps sur un ordinateur.
- B) 1974 : Un composant du système ne doit pas pouvoir effectuer une activité qui mette en péril le système principal.
- B) 2011 : Un système virtualisé ne doit pas compromettre le système hôte, soit directement (par exemple : perméabilité), soit indirectement (par exemple : surconsommation de ressources)
- C) 1974 : Un élément ou composant hors de confiance ("trust") ne devrait pas pouvoir être exécuté comme un élément de confiance.
- C) 2011 : Aujourd'hui, un tel principe serait traduit par le fait que le logiciel tournant sous forme virutalisée doit être considéré comme de confiance (Ex : Signé, testé, etc.) .
- D) 1974 : Il n'est pas possible de créer l'illusion d'un hardware différent (plus/moins de cpu, de mémoire, d'espace de stoquage, etc) sans impacter gravement les performances.
- D) 2011: Il n'est toujours pas possible de faire en sorte de simuler un matériel inexistant aussi rapidement que le même matériel réel.
- E) 1974 : Il est impératif de définir les besoins qui mènent au choix de la virtualisation et à son type d'implémentation avant de choisir de faire de la virtualisation.
- E) 2011 : Cette phrase n'a aujourd'hui pas besoin d'être adaptée.

Il est clair que même après plus de 40 ans, ces conseils et ces énoncés restent parfaitement d'actualité.

La suite de ce rappel historique va essayer de passer en revue une sélection des innombrables pas qu'a suivi la virtualisation dans l'histoire de l'informatique. Cette approche non exhaustive essaie de rester le plus possible dans la thématique de ce document et de la solution de virtualisation qu'il présente.

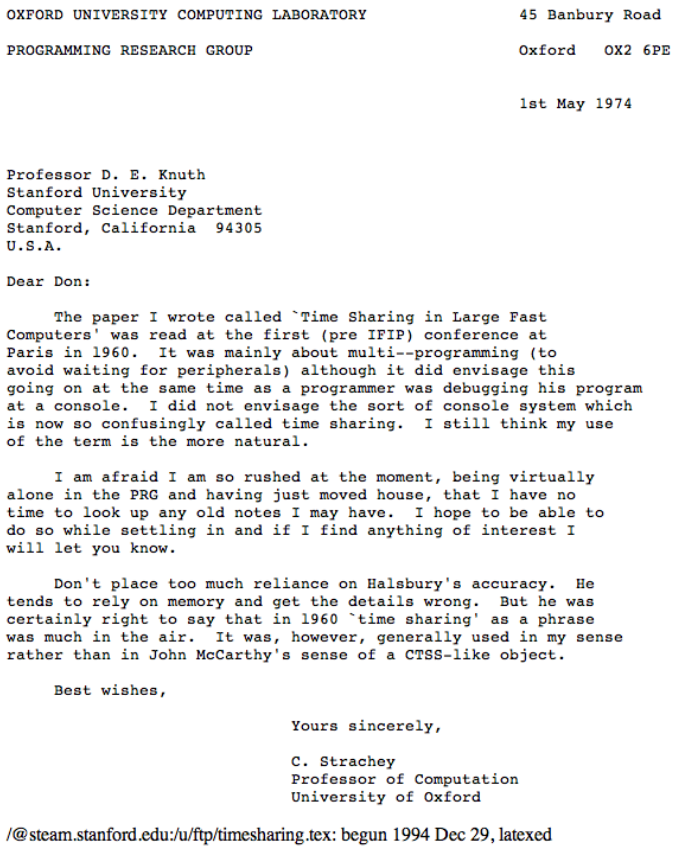
1.2 Point de vue académique

En juin 1955 Christopher Strachey [21] publiait à l'"International Conference on Information Processing at UNESCO" de New York une publication nommée "Time Sharing in Large Fast Computers" [22].

`Time Sharing ou, temps partagé`
Le temps partagé est une approche permettant de simuler le partage par plusieurs utilisateurs de temps processeur. Il ne faut pas le confondre avec le terme de multitâche : un système peut être multitâche sans être à temps partagé. [23]

Cette publication qu'il clarifiera plus tard en 1974 dans sa correspondance avec Donald Knuth [24] est ce qu'il est possible de considérer aujourd'hui comme la base historique de la virtualisation. Pour marquer cet historique, la figure suivante (Figure 1.1) extraite des archives de Stanford [25] montre un extrait de cette correspondance.

21. Article Wikipédia à propos de Christopher Strachey : http://en.wikipedia.org/wiki/Christopher_Strachey
22. Pour les personnes intéressées, cet article est disponible dans les archive de l'université de OXFORD https://www.nationalarchives.gov.uk/a2a/records.aspx?cat=161-csac71180&cid=6-6#6-6
23. Article WIkipédia à propos du "time sharing" : http://en.wikipedia.org/wiki/Time-sharing
24. Article wikipedia à propos de Donald Knuth : http://en.wikipedia.org/wiki/Donald_Knuth
25. http://www-formal.stanford.edu/jmc/history/timesharing/timesharing.html



OXFORD UNIVERSITY COMPUTING LABORATORY
45 Banbury Road

PROGRAMMING RESEARCH GROUP
Oxford OX2 6PE

1st May 1974

Professor D. E. Knuth
Stanford University
Computer Science Department
Stanford, California 94305
U.S.A.

Dear Don:

The paper I wrote called `Time Sharing in Large Fast Computers' was read at the first (pre IFIP) conference at Paris in 1960. It was mainly about multi--programming (to avoid waiting for peripherals) although it did envisage this going on at the same time as a programmer was debugging his program at a console. I did not envisage the sort of console system which is now so confusingly called time sharing. I still think my use of the term is the more natural.

I am afraid I am so rushed at the moment, being virtually alone in the PRG and having just moved house, that I have no time to look up any old notes I may have. I hope to be able to do so while settling in and if I find anything of interest I will let you know.

Don't place too much reliance on Halsbury's accuracy. He tends to rely on memory and get the details wrong. But he was certainly right to say that in 1960 `time sharing' as a phrase was much in the air. It was, however, generally used in my sense rather than in John McCarthy's sense of a CTSS-like object.

Best wishes,

Yours sincerely,

C. Strachey
Professor of Computation
University of Oxford

/@steam.stanford.edu:/u/ftp/timesharing.tex: begun 1994 Dec 29, latexed

FIGURE 1.1: Lettre de C. Strachey à Donald Knuth.

Dans cette lettre, la citation suivante est remarquable et mérite que l'on y prête attention :

> *« C. Strachey ... [my paper] was mainly about multi-programming (to avoid waiting for peripherals) although it did envisage this going on at the same time as a programmer who was debugging his program at a console. I did not envisage the sort of console system which is now so confusingly called time sharing." »*

Cela montre que même en 1960, l'expression "time sharing" était déjà à la mode et portait à confusion.

Une autre notion historique importante est celle du "Multiprogramming".

Multiprogramming ou multitâches
La notion de "multiprogramming" ou multitâches représente l'allocation d'un ordinateur et de ses ressources à plus d'une application concurrentes. Initialement, cette technique a été créée pour optimiser l'utilisation des ressources fournies par un ordinateur. Il s'agissait à l'époque d'empêcher une tâche en attente de bloquer toutes les autres. [26]

La première grande utilisation du "multiprogramming" pour distribuer des tâches informatiques peut être raccrochée à l'ordinateur Atlas [27] [28] au début des années 1960. Le projet Atlas était un projet mené conjointement par l'université de Manchester et la compagnie Ferranti [29].
Atlas a aidé à développer d'importants concepts de l'informatique moderne. En effet, il utilisait à l'époque un système de pagination [30] et des appels à un élément de supervision basique (Voir le chapitre Notion d'hyperviseur qui décrit ce qu'est un hyperviseur) appelé "extracode" en plus d'un système évolué de mise en attente

26. Article Wikipédia à propos du "multi-programming" : http://en.wikipedia.org/wiki/Multiprogramming
27. Article Wikipédia à propos de l'ordinateur Atlas : http://en.wikipedia.org/wiki/Atlas_Computer_(Manchester)
28. Article détaillé à propos de Atlas : http://www.chilton-computing.org.uk/acl/technology/atlas/p019.htm
29. Article Wikipédia à propos de la compagnie Ferranti : http://en.wikipedia.org/wiki/Ferranti
30. Article Wikipédia à propos du paging : http://en.wikipedia.org/wiki/Paging

FIGURE 1.2: Ordinateur Atlas - http://siarchives.si.edu/

des tâches ("spooling" en anglais).

Si l'on regarde des documentations écrites par les designers d'Altas en 1961, on peut remarquer des références à des paradigmes compris dans le domaine de la virtualisation. Par exemple, la citation suivante :

> *« Livre : "Classic operating systems: from batch processing to distributed systems" ISBN-13: 978-0387951133 ...the Supervisor extracode routines (S.E.R.'s) formed the principal 'branches' of the supervisor program. They are activated either by interrupt routines or by extracode instructions occurring in an object program. »*

Cette citation montre qu'Atlas utilisait une machine virtuelle pour le superviseur("hypervisor program") et une autre pour les programmes utilisateurs("object program").

Cette introduction montre que certains des balbutiements de la virtualisation ont commencé dans le milieu académique.

La partie suivante de ce chapitre présentera l'évolution dans le monde professionnel et le grand rôle de IBM et du MIT dans cette tâche.

1.3 Point de vue de l'informatique professionnelle

Dans le milieu des années 1960 au "IBM Warson Reseach Center" le projet M44/44X[31] naissait. Le but principal de ce projet était d'évaluer les concepts émergents de "time sharing". L'architecture de base de ce système était créée autour de plusieurs machines virtuelles :

- La machine principale (IBM 7044 (M44)).
- Chaque machine virtuelle était une image(clone) de la machine principale (ces images étaient appelées 44x).

L'espace mémoire utilisé par les 44x résidait dans la mémoire du M44. Ce système implémentait déjà à l'époque la notion de mémoire hiérarchique, de mémoire virtuelle et de "multi-programming".

`Virtual memory ou mémoire virtuelle`
En informatique, le mécanisme de mémoire virtuelle a été mis au point dans les années 1960 afin de permettre

31. Article Wikipédia à propos du projet M44/44X : http://en.wikipedia.org/wiki/IBM_M44/44X

à des programmes de pouvoir s'exécuter dans un environnement matériel possédant moins de mémoire centrale que nécessaire (ou, vu autrement, de faire tourner plus de programmes que la mémoire centrale ne peut en contenir). [32]

Précédemment au développement des M44/44x, IBM avait fourni une série d'ordinateurs (IBM 704) et ses évolutions (709,7090,7094) à des ingénieurs du MIT dans les années 1950. Ce sont sur ces ordinateurs que le MIT a développé un hyperviseur (Compatible Time Sharing System (CTSS) [33]) qui a fait apparaître certains des concepts les plus importants de la virtualisation et des systèmes d'exploitation modernes, comme par exemple :

- Supervision des I/O en lieu et place des programmes (I/O réalisée par le système et non directement par les programmes).
- "Scheduling" des tâches frontales et des tâches de fond.
- Stockage temporaire des programmes en attente d'exécution et swap de mémoire (voir figure 1.3).
- etc..

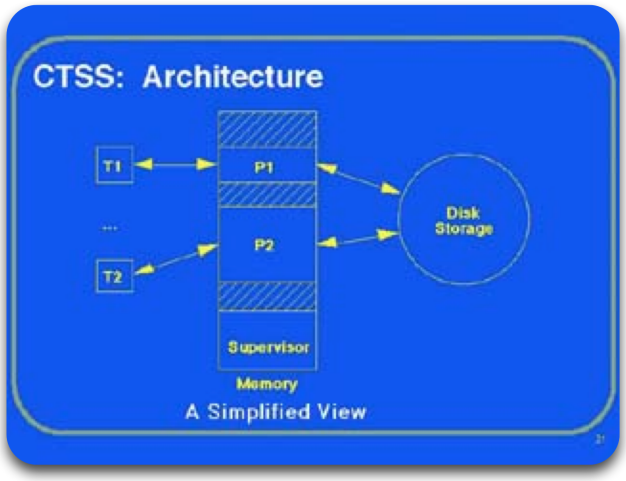


FIGURE 1.3: Premier schéma de la mémoire du CTSS et de son hyperviseur - http://larch-www.lcs.mit.edu:8001/corbato/turing91/

A la même époque, IBM commençait les premières conceptions des célèbres ordinateurs de la famille des 360 [34]. Ces ordinateurs allaient devenir les premiers grands éléments d'infrastructures de virtualisation commercialisés.

En 1963, au MIT, le projet MAC [35] était créé. Ce projet de très grande ampleur sera même à terme directement fusionné avec le "MIT Laboratory for Computer Science". L'objectif principal de ce travail était d'améliorer l'implémentation de CTSS qui était originellement développé en partenariat avec IBM.

Ce grand projet aboutira sur Multics [36] [37] qui sera une référence en termes de "time-sharing" et de modèle de conception pour les systèmes d'exploitation jusqu'aux systèmes modernes que nous utilisons actuellement. La machine Multics sera aussi la première à apporter le concept de niveaux de privilèges matériel aux processeur (Voir : le sous chapitre d'introduction : Une problématique global et la figure 1.4). Cependant, contrairement aux années 1950, l'offre de partenariat sera obtenue par General Electric's [38](monstre économique

32. Article Wikipédia à propos du principe de mémoire virtuelle : http://en.wikipedia.org/wiki/Virtual_memory
33. Article Wikipédia à propos de CTSS : http://en.wikipedia.org/wiki/Compatible_Time-Sharing_System
34. Voir le chapitre : intro
35. Article Wikipédia à propos du projet MAC : http://en.wikipedia.org/wiki/Project_MAC#Project_MAC
36. Article Wikipédia à propos de Multics : http://en.wikipedia.org/wiki/Multics
37. Article complet et détaillé à propos de Multics : http://www.multicians.org/multics-vm.html
38. Article Wikipédia à propos de General Electric : http://en.wikipedia.org/wiki/General_Electric

de l'époque où les firmes de production d'énergie et de télécommunication étaient les leaders technologiques incontestés).

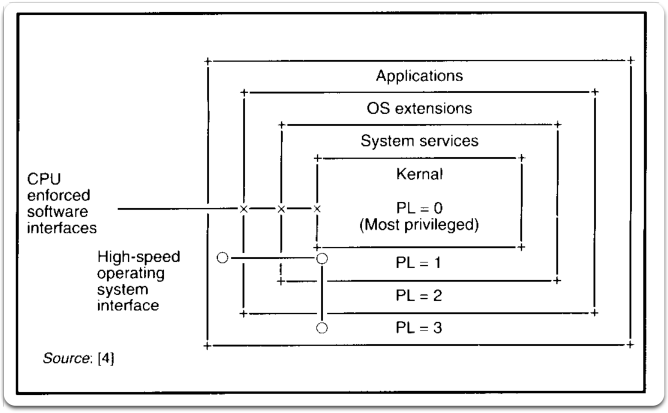


FIGURE 1.4: Architecture processeur avec niveaux de privilège de Multics - Information Management & Computer Security ISSN: 0986-5227

Malgré cet échec, IBM est resté l'innovateur le plus important dans ce domaine (et reste à ce jour un des leaders dans le monde de l'innovation technologique). Dans les années qui suivent et jusqu'à nos jours, IBM commercialisera beaucoup de solutions de virtualisation créant avec cela la plupart des paradigmes modernes de la virtualisation (comme par exemple, le principe de la paravirtualisation (Voir le chapitre Paravirtualisation et virtualisation complète)).

Globalement, la philosophie de IBM restera identique à celle utilisée dans les M44/44x. Beaucoup de machines peuvent être créées sur un même hyperviseur, mais toutes restent à l'origine des copies d'un modèle identique.

Plus les technologies évolueront, plus les Hypverviseurs logiciels pourront se retrouver proches du matériel et changeront de nom pour être appelé VMM(Virutal Machine Monitor). Avec les VM/370 [39] qui deviendront plus tard les ordinateurs de la séries Z (aujourd'hui l'une des solutions de mainframe les plus utilisées), IBM inaugurait l'un des premiers VMM/Hyperviseur capable de fonctionner avec de l'aide provenant d'un matériel dédié (aujourd'hui cette situation est celle proposée avec les instructions de virtualisation sur processeurs X86 [40] [41]).

De nos jours, ces solutions de mainframe sont utilisées comme élément central dans beaucoup d'institutions (banque, assurances, énergie, etc). Cependant, IBM regarde déjà vers l'avenir avec ces solutions de mainframe X86 appelée System-X [42]. Ces solutions, qui sont justement basées sur la technologie open-source KVM (Figure 1.5 [43] présente ce passage) traitée dans ce travail d'approfondissement.

39. Article Wikipédia sur les ordinateurs de la série VM : http://en.wikipedia.org/wiki/VM_(operating_system)
40. article Wikipédia à propos de la virtualisation X86 : http://en.wikipedia.org/wiki/X86_virtualization
41. Voir paragraphe concernant l'introduction à ces instructions dans le chapitre XX concernant la notion d'hyperviseur
42. http://www-03.ibm.com/systems/x/
43. Présentation téléchargeable ici :https://www.redhat.com/promo/summit/2010/presentations/summit/decoding-the-code/thurs/ghuizenga-200/NEW-Virtualization-KVM-at-IBM-Posting-version.pdf

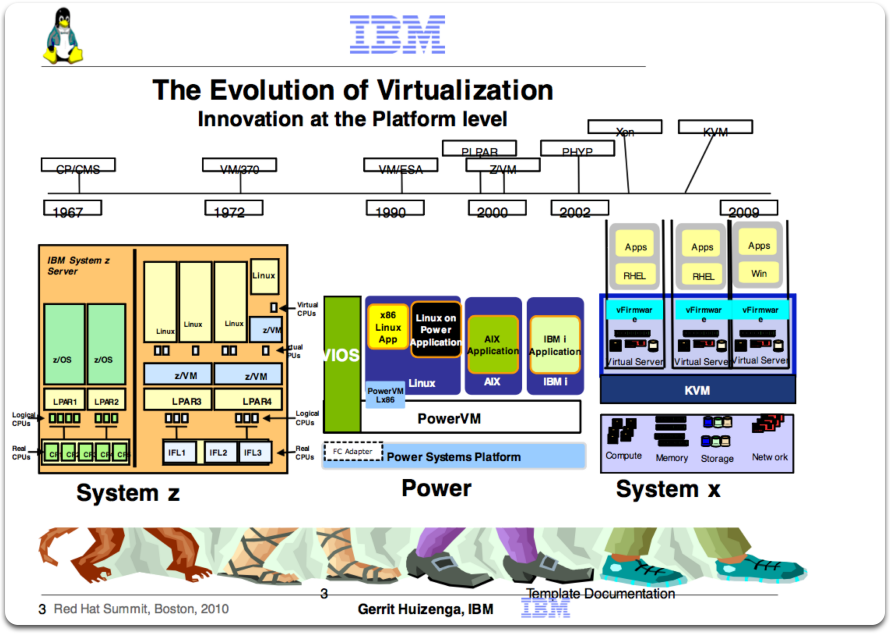


FIGURE 1.5: Slide extraite d'une présentation données par IBM 1.5 dans une conférence RedHat montrant le choix de IBM d'aller vers KVM

1.4 Conclusion

Ce chapitre a présenté de manière très raccourcie, au travers de thématiques choisies, l'évolution de la notion de virtualisation dans le temps rattachant ses premiers pas à son avenir proche (KVM). Les chapitres suivants vont essayer de présenter certains concepts clés du domaine de la virtualisation permettant au lecteur de se préparer à la compréhension de l'architecture de KVM.

Les lecteurs intéressés pourront trouver sur Wikiédia les dates importantes (l'article est relativement complet même si il manque encore certaines références importantes de la virtualisation [44]). Cet article comporte aussi de très bonnes références en bas de page.

44. Dates importante de la virtualisation : http://en.wikipedia.org/wiki/Timeline_of_virtualization_development

Sommaire

Chapitre

2

Notions d'hyperviseur et de virtualisation

Painting is an illusion, a piece of magic, so what you see is not what you see.
Philip Guston.

Dans la notion de virtualisation actuelle, un hyperviseur est l'une des techniques de virtualisation qui permet de faire fonctionner plusieurs systèmes d'exploitation sur un même hardware. Cette technologie est souvent appelée "hardware virtualization", "platform virtualization" ou encore "VMM (Virtual Machine Monitor) dans la littérature disponible. La première partie de chapitre présente les bases conceptuelles de ce qu'est un hyperviseur et comment il est généralement conçu.

De manière générale, le but d'un hyperviseur est de présenter à un système invité une plateforme matérielle compatible afin qu'il soit possible pour l'invité de s'exécuter dessus. Le terme "hyperviseur" est apparu la première fois en 1965 comme un élément logiciel accompagnant les IBM 360/65. En 1970, PR/SM [1] était l'hyperviseur le plus présent dans les mainframes IBM, et encore aujourd'hui, cet hyperviseur est utilisé dans de nombreuses infrastructures de production.

La fin de ce chapitre essaiera de présenter de manière simplifiée certains concepts fondamentaux choisis. Les informations proposées dans ce chapitre ont volontairement été sélectionnées afin de préparer le lecteur à la compréhension de KVM.

1. Article de IBM à propos de leurs PR/SM : http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp?topic=\%2Feicaz\%2Feicazzlpar.htm

2.1 Catégories d'hyperviseur

Si l'on s'accorde avec la définition de Robert P.Goldberg [2] qui figure dans sa publication titrée "Architectural Principles for Virtual Computer Systems" [3], deux types d'hyperviseurs sont définissables (Voir figure 2.1).

- **Les hyperviseurs de type 1 :** Ou, souvent appelés "native" et "bare metal". Ces hyperviseurs fonctionnent directement sur le hardware (mais ne sont pas inclus complètement dans le hardware). Ce modèle représente le modèle théorique historiquement idéal de machine virtuelle. C'est le modèle qui avait été choisi par les IBM CP/CMS [4] ou encore Z/VM. Mais, aujourd'hui, VMware ESXi, Citrix XenServer, KVM ou encore WINE (logiciel d'émulation) [5] peuvent aussi être considérés comme des hyperviseurs de type 1.
- **Les hyperviseurs de type 2 :** Ou, souvent appelé "Hosted". Ces hyperviseurs sont des éléments logiciels qui fonctionnent sur un système d'exploitation existant. Des exemples modernes pourraient être VirtualBox, VMware workstation, WMware server.

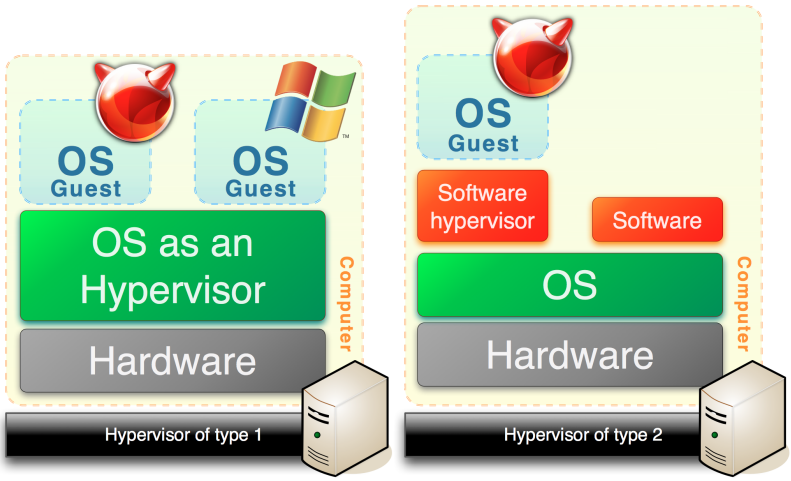


FIGURE 2.1: De gauche à droite, hyperviseur de type 1 et de type 2

La figure 2.1 présente les deux types d'hyperviseurs de manière volontairement simplifiée. A ce stade, il est important de saisir la notion de couche entre les différents niveaux d'abstraction. La figure 2.1 présente donc un hyperviseur comme une couche/interface logicielle qui propose un service de virtualisation/mutualisation du matériel de la machine sur laquelle il est installé. Ce support peut être ou non aidé par du matériel facilitant cette virtualisation/mutualisation. Chaque machine virtuelle (OS Guest sur 2.1) voit donc le matériel que l'hyperviseur lui présente.

La suite de ce chapitre s'intéresse à l'architecture d'un hyperviseur et à comment le représenter de manière simple.

2.2 Anatomie d'un hyperviseur

Comme expliqué précédemment, un hyperviseur est une couche logicielle qui permet la mutualisation/virtualisation/abstraction d'une couche matérielle entre plusieurs logiciels (machines virtuelles dans notre cas) en ayant besoin. D'un point de vue simplifié, un hyperviseur est donc la "colle" qui permet aux systèmes invités de fonctionner de manière concurrente en même temps que le système hôte sur un matériel donné.

2. Voir chapitre : La virtualisation, une problématique globale et historique
3. Pour les intéressés, la publication est disponible dans les archives de l'université de Yale : http://flint.cs.yale.edu/cs428/doc/goldberg.pdf ainsi qu'une autre version dans les archives du NTIS : http://www.dtic.mil/cgi-bin/GetTRDoc?AD=AD772809&Location=U2&doc=GetTRDoc.pdf
4. Point de vue de l'informatique professionnelle
5. Site officiel de WINE : http://www.winehq.org/

Présentons à présent comment, et avec qui, l'hyperviseur réalise son travail. Afin de réaliser la tâche qui est la sienne (d'un point de vue générique), un hyperviseur n'a besoin que de peu de choses pour permettre au système invité de démarrer (voir figure 2.2) :

- Une image du système à démarrer. Par exemple, un noyau Linux.
- Quelques éléments de configuration (IP, mémoire allouée, nombre de cpu visibles, etc.).
- Un disque virtuel (espace de stockage).
- Une interface réseau.

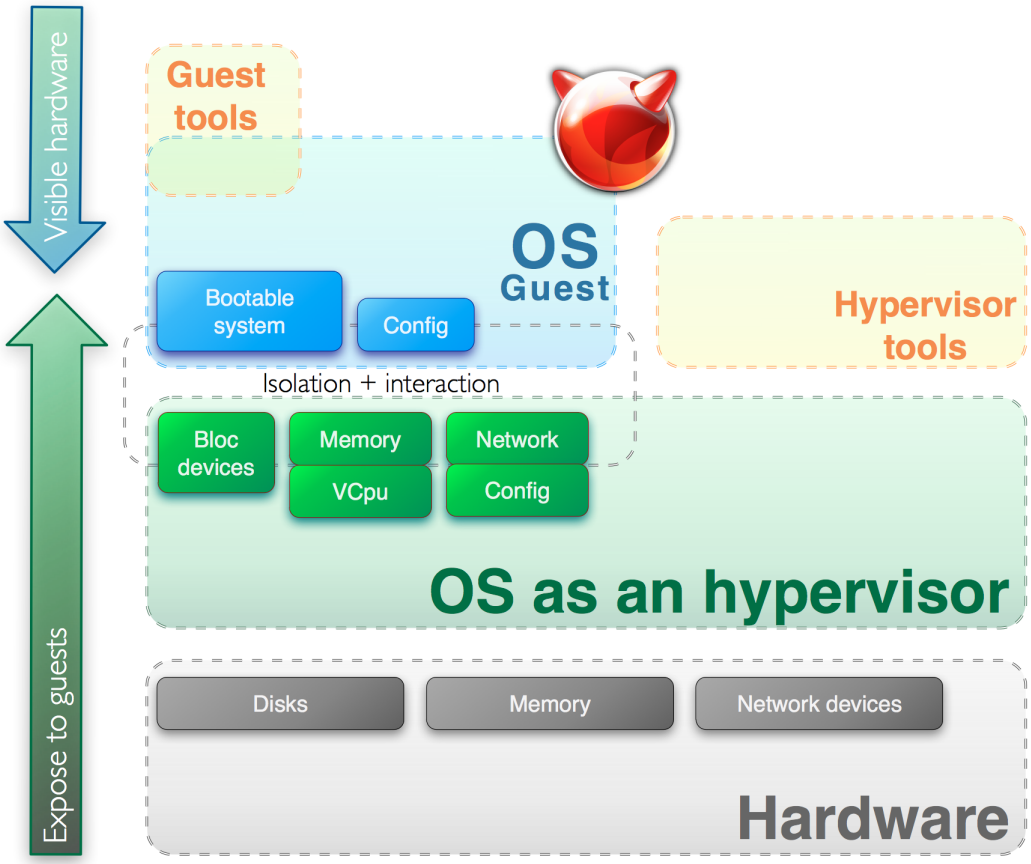


FIGURE 2.2: Éléments minimaux d'un hyperviseur

En général, les interfaces réseaux et les disques virtuels sont des "mappages" réalisés avec du matériel existant. Un dernier élément important sont les utilitaires fournis par les développeurs de la solution de virtualisation pour gérer de manière efficace les machines virtuelles. Communément, ces outils sont appelés "Guest Tools" pour ceux qui sont installés dans les systèmes virtualisés et "hypervisor tools" pour ceux installés sur le système hôte ou permettant d'interagir avec l'hyperviseur

Pour réaliser ses fonctionnalités, un hyperviseur a besoin de composants simples mais cruciaux. La figure 2.3 présente de manière un peu plus détaillées les éléments principaux d'un hyperviseur.

En général, les espaces noyau et utilisateur d'un système d'exploitation communiquent entre eux à l'aide d'appels système (utilisateur -> noyau). Il existe un principe équivalent lors de la communication entre la machine

virtuelle et son hôte. Ces communications sont appelées "hypercall" ou encore "hypercall layer". Elles sont très dépendantes de l'implémentation choisie mais ce principe est présent dans toutes les solutions de virtualisation.

En ce qui concerne les "I/O", elles peuvent être complètement virtualisées dans le système hôte et/ou assistées par du code (assistance proposée par les cpu modernes) au niveau des systèmes invités.

Le cas des interruptions [6] est un peu différent. Les interruptions dans l'état actuel des hyperviseurs doivent forcément être complètement traitées par celui-ci. Le système hôte hébergeant l'hyperviseur doit pouvoir traiter ces interruptions en plus des interruptions qui lui sont propres. Les interruptions concernant des hôtes virtualisés sont simplement redirigées par le système vers le périphérique virtuel correspondant. De la même façon, un hyperviseur interceptera les "traps" ou les "Exceptions" concernant un hôte virtuel. Il est nécessaire que si un "guest" crash ou se fait arrêter, l'hyperviseur et les autres systèmes virtualisés ne soient pas affectés.

Un élément fondamental est le gestionnaire de pages mémoires. Classiquement, le rôle de ce composant est de permettre au système d'adresser des pages mémoires (depuis l'hyperviseur ou un hôte virtuel) se trouvant dans la mémoire ou sur un matériel disponible dans l'espace adressable. Ce composant est un élément clé dans l'isolation des systèmes. Un système virtuel ne devrait en aucun cas pouvoir accéder à des zones mémoires qui ne lui sont pas autorisées.

Le dernier élément clé d'un hyperviseur est le scheduler, sont rôle est de gérer la répartition du temps cpu entre les différentes tâches d'un système. Dans le cas d'un hyperviseur, le scheduler prend aussi la responsabilité de répartir les ressources entre le système et les hôtes virtualisés.

La figure suivante (Fig 2.3) présente ces différents éléments sous forme illustrée.

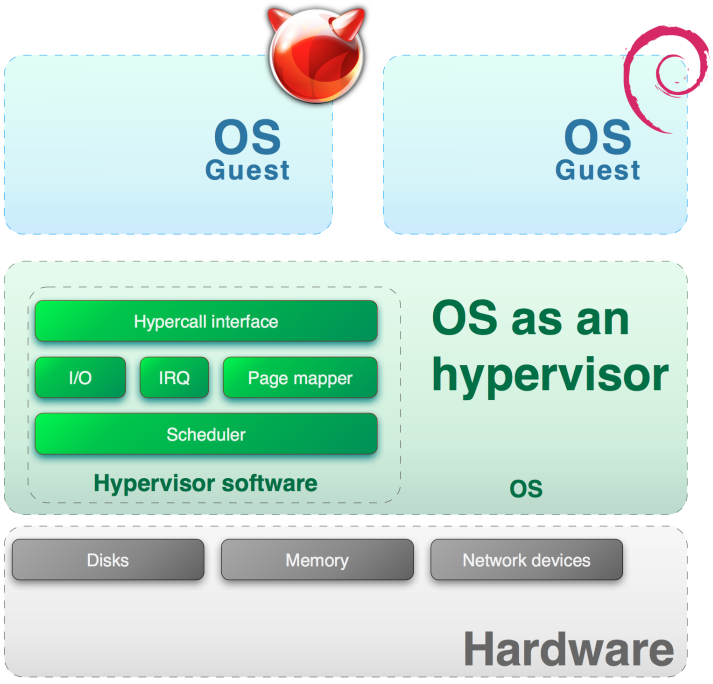


FIGURE 2.3: Éléments de base d'un sytème d'hyperviseur

6. Article Wikipédia à propos des interruptions : http://en.wikipedia.org/wiki/Interrupt

2.3 Introduction à la virtualisation et la paravirtualisation

Comme les sections précédentes l'ont montré, la virtualisation permet le partage de ressources matérielles et une utilisation optimale de celles-ci. En général, la notion de matériel englobe plus que simplement un processeur. Cela inclut aussi d'autres éléments, comme les cartes réseaux, des disques, etc. Certains éléments matériels peuvent être virtualisés simplement, comme par exemple le processeur ou les disques, mais, d'autres, comme une carte graphique ou un port série, sont plus durs à virtualiser car plus difficiles à partager entre plusieurs utilisations concurrentes.

Cette section essaie d'expliquer de manière globale les différentes manières de réaliser ce partage et présentera la direction que prennent les futures solutions de virtualisation.

2.3.1 De l'émulation de périphérique à la paravirtualisation classique

La solution la plus simple pour gérer du matériel est d'utiliser l'émulation. L'émulation permet de créer un composant purement logiciel qui sera exécuté à la place du matériel et qui, par la suite, utilisera le matériel. Le système pourra donc gérer ces différents composants comme de simples processus demandant un accès à des fonctionnalités du système liés aux périphériques matériels désirés.

Il existe deux grandes méthodes dans l'émulation de matériel: l'émulation dans l'hyperviseur et l'émulation en espace utilisateur.

L'émulation dans l'hyperviseur est une solution qui consiste à placer la gestion des éléments virtualisés à l'intérieur de l'hyperviseur. De cette manière, ses différents composants (disques virtuels, cartes réseaux virtuelles, etc.) sont gérés dans le système de l'hyperviseur. La figure suivante 2.4 présente cette architecture.

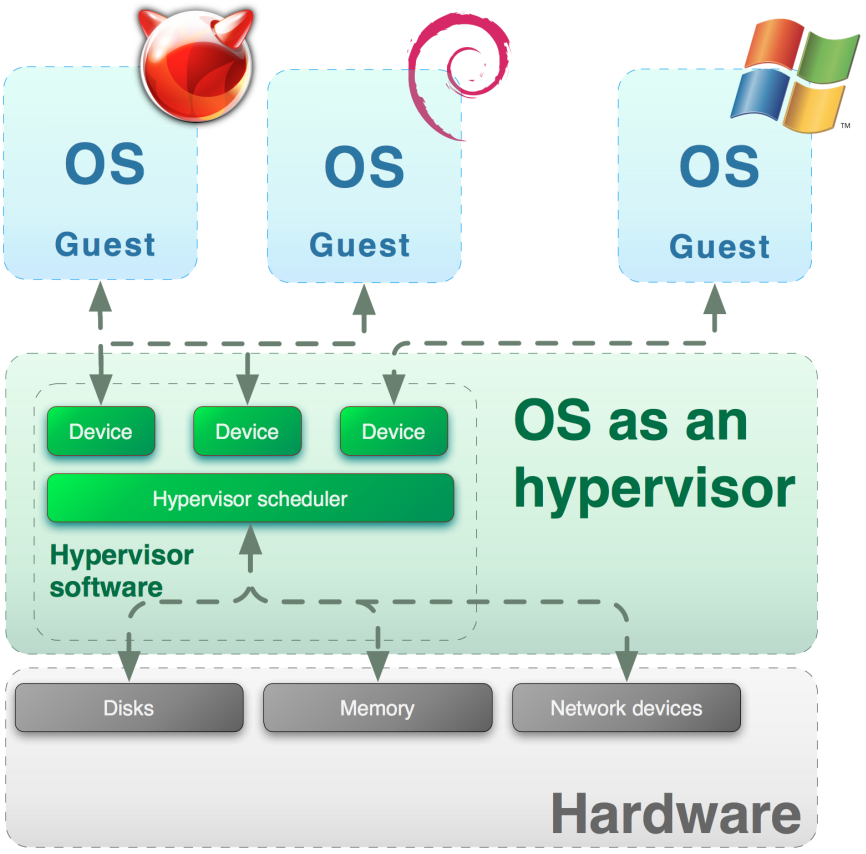


FIGURE 2.4: Virtualisation de périphérique dans l'hyperviseur

Cette figure montre bien que certains périphériques virtuels peuvent être partagés entre plusieurs machines virtuelles .

Le second type d'architecture est appelé "userspace device emulation" (2.5). Cette option implique d'utiliser des périphériques virtuels se trouvant en espace utilisateur. Ce modèle apporte beaucoup d'avantages, comme par exemple la séparation de la virtualisation des périphériques du fonctionnement de l'hyperviseur, le fonctionnement des périphériques avec des privilèges réduit ou encore sa gestion, qui devient la charge du système et plus de l'hyperviseur lui-même.

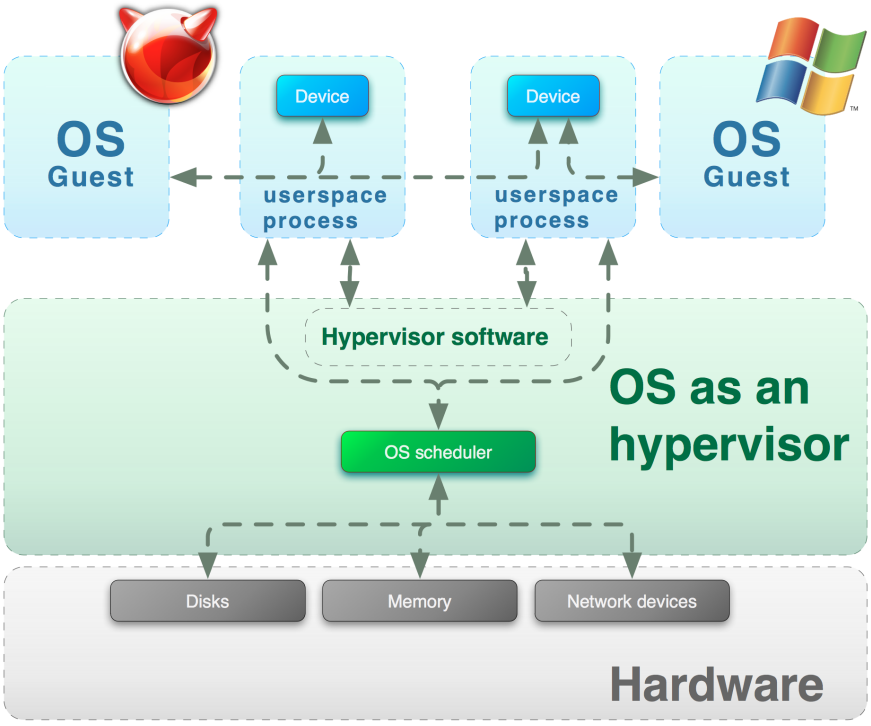


FIGURE 2.5: Virtualisation de périphérique dans l'espace utilisateur

Cependant, cette approche comporte aussi des inconvénients comme par exemple: l'hyperviseur a moins de contrôle sur les périphériques virtualisés et le "trusted computing" [7] devient plus difficile.

En plus de ces modèles de simulation, il existe aussi le principe de paravirutalisation qui dans sa définition fondamentale permet de rendre plus efficace les deux modes précédemment décris.

Des drivers paravirtualisés sont des drivers qui comportent une partie du périphérique virtuel dans le système invité et une autre dans le périphérique virtualisé maîtrisé par l'hyperviseur. Cette solution permet d'optimiser le transfert de données entre ces deux mondes, d'améliorer la gestion de la concurrence ou encore d'adapter la façon dont un périphérique répond en fonction du matériel présent sur le système hôte.

2.3.2 Le "Device passthrough"

Aujourd'hui, les technologies de paravirtualisation évoluent de plus en plus. Il existe un concept qui est souvent implémenté dans les drivers paravirtualisés mais qui est un principe en lui même, le "passthourgh".

Que la simulation d'un périphérique soit effectuée dans l'hyperviseur, dans l'espace utilisateur ou en partie dans le système invité, il existe toujours un overhead de traitement entre l'utilisation du matériel virtuel et l'utilisation du matériel réel. Il existe cependant des moyens d'améliorer grandement ce fait.

7. Article Wikipédia à propos du "trusted computing" : http://en.wikipedia.org/wiki/Trusted_Computing

Dans le cas où le partage de ressources devient un point négociable, il est possible d'optimiser grandement la situation. D'un point de vue général, l'idée derrière le "passthrourgh" est de donner en partie, ou complètement, l'exclusivité d'un périphérique définitivement, ou pour un temps donné, à un hôte virtuel. Ce procédé a par exemple lieu quand un processeur, à l'aide des assistances matérielles, permet à un système invité de s'exécuter nativement dessus pour un certain temps. Mais il est aussi possible d'imaginer allouer, par exemple, une carte réseau ou un port usb à un système invité de manière définitive.

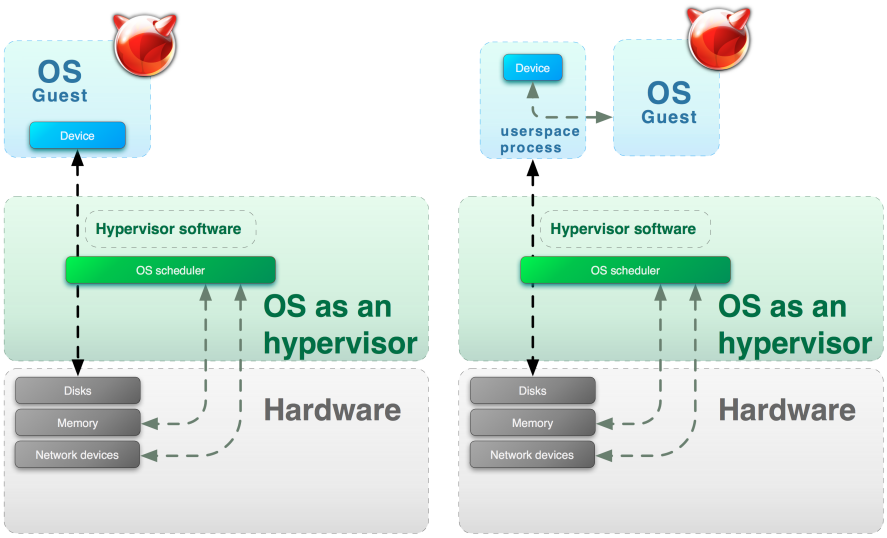


FIGURE 2.6: Virtualisation de périphérique avec le passthrough

Cette solution permet dans les cas où elle est applicable de s'affranchir du travail des couches intermédiaires qui doivent traiter chaque demande et l'accès au périphérique virtuel pour de multiples systèmes.

Aujourd'hui, les drivers dit de paravirutalisaion ont tendance à utiliser tous les moyens possibles pour offrir au maximum une exclusivité temporaire du matériel aux systèmes invités quand cela est possible. C'est dans ce sens que l'assistance matérielle à la virtualisation se dirige.

Ces solutions ont bien évidement des désavantages comme les limitations en cas de migration d'un hyperviseur à un autre. Dans ces cas, il faut par exemple être capable de réaffecter un autre périphérique complètement dédié au système invité sur le nouvel hyperviseur.

Les solutions du futur se dirigent donc vers de plus en plus de matériel capable de gérer la virtualisation, voir de systèmes hardware (carte pci/pci-express) disposant d'unités de calcul pour soulager la gestion de certains périphériques virutels (par exemple : une carte d'extention capable d'être un grand nombre de processeurs graphiques virtuels).

2.4 Conclusion

Ce chapitre a montré de manière courte un tour d'horizon rapide des technologies de virtualisation. Le temps pour ce travail étant limité il n'a malheureusement pas été possible de tout présenter comme par exemple : le rôle des nouvelles instructions VT ou les nombreuses améliorations hardware disponibles dans le monde de la virtualisation. Les personnes intéressées trouveront dans les liens suivants [8] [9] de la documentation à propos des assistances matérielles à la virtualisation

8. Intel Virtualization Technology : http://www.intel.com/technology/itj/2006/v10i3/2-io/5-platform-hardware-support.htm

9. Hardware Virtualization Rootkits (contient une bonne introduction aux instructions VMX) : http://www.theta44.org/software/HVM_Rootkits_ddz_bh-usa-06.pdf

Cependant, les notions minimales nécessaires pour commencer l'étude de système complexe comme KVM ont été expliquées dans cette partie de document.

La suite du document s'intéresse à différentes solutions de virtualisation, pas forcément très connues, mais qui proposent chacune des aspects technologiques intéressants.

Chapitre

3

Tour d'horizon des solutions de virtualisation

In wisdom gathered over time I have found that every experience is a form of exploration.

Ansel Adams.

Sommaire

Le monde de la virtualisation est un monde large occupé par beaucoup de solutions concurrentes. Souvent, lorsque l'on évoque le domaine de la virtualisation seule une petite partie de ces acteurs viennent en tête. Il est cependant intéressant de ce rendre compte de la variété de solutions présentes dans ce domaine.

Les chapitres précédents ont montré que le sens traditionnel de la virtualisation était de permettre à plusieurs environnements d'exécution (Machine virtuel) de fonctionner sur un même hardware. Chacune de ces machines virtuelles doit sembler identique à du matériel réel pour ces utilisateurs alors qu'en réalité, elles se trouvent dans un environnement cloisonné et contrôlé. Historiquement, les concepts d'émulation et de virtualisation ont toujours été mélangés de par le but commun qu'ils poursuivent. Ce chapitre se consacre à passer en revue une série de "technologies, méthodologies, projets, produits et concepts" liés directement et indirectement à la virtualisation. Par technologies de virtualisation il est entendu des moteurs de virtualisation et non des produits utilisant des moteurs de virtualisation. Tous ne seront bien sûr pas présents, un choix a dû être réalisé. Certains éléments choisis sont d'excellents outils pédagogique dont l'étude du code open-source pourrait être une formation à la virtualisation en soit, d'autres parce qu'ils sont des acteurs fréquemment cités, etc.

Tous ces systèmes ne seront bien sûr pas expliqués de manière détaillée mais, le lecteur intéressé pourra utiliser les nombreuses références fournies pour compléter les informations présentées.

3.1 Présentation de solutions choisies

3.1.1 Émulation ABI/API

Plutôt que de créer simplement des machines virtuelles qui font fonctionner un système entier, l'émulation d'API peut être utilisée pour créer des environnements cloisonnés permettant l'exécution de programmes extérieurs sur une plateforme données.

Le cas de SUN

Sun par le passé a utilisé WABI (Windows applications Windows Calls)[1] pour permettre à Solaris[2] d'utiliser dans une certaine mesure des applications Windows. WABI est conçu comme une couche qui se situe entre les applications et le système d'exploitation qui intercepte les appels au système windows et les traduit dans leurs équivalents UNIX. Dans le monde x86, les instructions invitées (applications windows) sont directement exécutées sur le processeur, dans le cas du monde SPARC, les instructions sont soit émulées soit traduites quand cela est possible.

Plus tard, Sun proposa aussi une solution d'émulation d'environnement PC sur Solaris équipé de processeur SPARC. Cette solution se nommait SunPC et proposait l'émulation d'un processeur x86 286. Le point intéressant avec cette solution et qu'elle était fréquemment utilisée conjointement à une carte de type SBC (Single Card Computer)[3] et permettait[4] l'expérience d'utilisation en exécutant de manière native certaines instructions.

Lxrun

Lxrun[5] est une application qui permet l'exécution de code ELF[6] (Exécutable Linux) x86 sur d'autres systèmes compatibles x86 mais, non-linux comme par exemple : SCO OpenServer[7], SCO UnixWare[8], Solaric etc.

Lxrun réalisait son travail en effectuant un "remapping" des appels système Linux vers les bons appels à la volée. De manière générale, Lxrun est donc un émulateur d'appel système.

LynxOS et FreeBSD

Il n'est certainement pas possible de citer toutes les technologies mais, par exemple, les nouvelles version du système temps-réel LynxOS[9] possèdent des capacités d'ABI.

De manière similaire, freeBSD permet une compatibilité avec les binaires Linux. Cependant, dans le cas de FreeBSD la situation a évolué, ce système proposant actuellement un "process file system"(procfs) qui permet l'émulation d'un sous ensemble du procfs de linux.

`Système de fichier processus (procfs) :`

Sur les systèmes du type Unix, procfs (process file system, système de fichiers processus en anglais) est un pseudo-système de fichiers (pseudo car dynamiquement généré au démarrage) utilisé pour accéder aux informations du noyau sur les processus. Le système de fichiers est souvent monté sur le répertoire /proc.[10]

Wine

Wine[11] est une solution permettant d'exécuter des applications Windows sous Linux, FreeBSD, Solaris, MAC OSX, Windows sans pour autant simuler la moindre instruction processeur. Wine repose sur un portage et une utilisation des librairies natives Windows sous Linux. Il s'agit donc dans le cas présent de quelque chose entre la simulation et la traduction à la volée des appels.

1. Article Wikipédia à propos de SUN WABI : http://en.wikipedia.org/wiki/Wabi_(software)
2. Article Wikipédia à propos de l'os Oracle Solaris : http://en.wikipedia.org/wiki/Solaris_(operating_system)
3. Article Wikipédia à propos des SBC : http://en.wikipedia.org/wiki/Single_Board_Computer
4. Artcile wikipédia à propos des cartes sunPci : http://en.wikipedia.org/wiki/SunPCi
5. Article Wikipédia à propos de Lxrun : http://en.wikipedia.org/wiki/Lxrun
6. Voir livre Linux Kernel Developpement
7. Article Wikipédia à propos de SCO OpenServer : http://en.wikipedia.org/wiki/SCO_OpenServer
8. Article Wikipédia à propos de SCO UnixWare : http://en.wikipedia.org/wiki/UnixWare
9. Article Wikipédia à propos de LynxOS : http://en.wikipedia.org/wiki/LynxOS
10. Article Wikipédia à propos des procfs : http://en.wikipedia.org/wiki/Procfs
11. Site officiel de Wine : http://www.winehq.org/

MainWin et anecdotes historiques entre Solaris et Microsoft

Il existe une version de Microsoft Internet Explorer et Outlook Express pour Solaris(SPARC). Ce qui est intéressant avec ces versions est qu'il ne s'agit pas de portages à proprement parler. Ces versions utilisent une émulation de l'API Windows pour fonctionner. Ces versions sont le travail de MainSoft [12] [13].

Aujourd'hui, MainSoft comercialise un produit nommé MainWin qui permet aux développeurs d'applications développées avec Visual Studio de faire fonctionner leurs applications avec Solaris, HP-UX, Linux, etc. MainWin recompile les applications depuis les sources utilisant des simulations d'API compatibles avec la plateforme cible.

3.1.2 Bochs

Bochs [14] [15], prononcé "bux" en anglais est un émulateur x86 open source écrit en C++. Purement en user-space, cet émulateur gère un processeur x86 avec quelques périphériques. De par son architecture légère, Bochs est facilement portable d'une plateforme à une autre et est maintenant disponible sur beaucoup de systèmes différents.

Lent, ce qui n'est pas surprenant vu que toutes les instructions et les périphériques sont émulés, Bochs n'en reste pas moins un outil extrêmement flexible et customizable en cas de besoins précis. De plus, son code claire et son architecture simple le rendent très efficace comme outil d'apprentissage aux principes de virtualisation.

3.1.3 Chorus

Le noyau du système Chorus [16] proposait un "framework" bas niveau qui permettait à des systèmes distribués d'être implémentés au dessus de lui. Par exemple, System V Unix [17] a été implémenté en dessus de Chorus. Pour réaliser ces portages, des instructions spécifiques d'émulation étaient utilisées dans le noyau de Chorus.

3.1.4 Chroot()

A l'origine, chroot [18] (change root) est une commande des systèmes d'exploitation UNIX permettant de changer le répertoire racine d'un processus de la machine hôte. De nombreux frameworks et plus particulièrement ceux à destination des fournisseurs de services reposent sur chroot. Ces environnements utilisent le plus souvent chroot pour créer un bac à sable pour systèmes de fichiers (filesystems sandboxing) que cela soit dans l'espace utilisateur ou l'espace noyau des systèmes utilisés. Par exemple, les "Jail"(voir Les jails BSD) de FreeBSD utilisent chroot.

Virtfs [19] est aussi un exemple de solution basée sur chroot pour Linux.

3.1.5 Denali

Denali [20] [21] ou "Denali isolation kernel" est un système d'exploitation proposant des fonctionnalités de superviseur/moniteur afin de permettre à des services non sûrs d'être isolés dans des "domaines" protégés. Denali utilise fortement la paravirtualisation afin de proposer des performances de haut niveau. Pour parvenir à ceci, tous les systèmes fonctionnant dans un système Denali doivent être modifiés (Kernel, drivers, etc) pour supporter le matériel paravirtualisé exposé par Denali. Par exemple, le processeur exposé par Denali n'est pas un processeur x86 mais, une version modifiée pour permettre une meilleure virtualisation et montée en charge.

12. Site officiel de MainSoft : http://www.mainsoft.com/content/about-mainsoft
13. Article Wikipédia à propos de MainSoft : http://en.wikipedia.org/wiki/Mainsoft
14. Article wikipédia à propos de Bochs : http://en.wikipedia.org/wiki/Bochs
15. Site officiel de Bochs : http://bochs.sourceforge.net/
16. Extrait du livre "Edition 2 of Distributed Systems" à propos de Chorus : http://www.cdk3.net/oss/Ed2/Chorus.pdf
17. Article Wikipédia à propos de System V Unix : http://en.wikipedia.org/wiki/UNIX_System_V
18. Article Wikipédia à propos de chroot : http://en.wikipedia.org/wiki/Chroot
19. Site officiel de Virtfs : http://www.prongs.org/virtfs/
20. Site officiel de Delani : http://denali.cs.washington.edu/
21. Publications scientifique sur les performances et l'architecture de Denali : http://denali.cs.washington.edu/pubs/distpubs/papers/denali_sigops.pdf http://denali.cs.washington.edu/pubs/distpubs/papers/denali_osdi.pdf http://denali.cs.washington.edu/pubs/distpubs/papers/denali_usenix2002.pdf

A l'origine, Denali était prévu pour faire fonctionner des machines extrêmement légères ne faisant tourner qu'une seule application. L'utilité de base de cette architecture a pour but de permettre à un fournisseur de proposer du "Software as a service" de manière modulaire et sécurisée. Les nouvelles versions de Delani ont été adaptées (nommées uDelani) pour supporter des systèmes d'exploitation complet en virtualisation.

3.1.6 Dis

Dis [22] est le système de virtualisation développé pour le système d'exploitation Inferno [23] du `Bell Labs`. L'architecture de Dis est très semblable à celle utilisée pour le langage Java à de très nombreux aspects. Les inventeurs de Dis décrivent les principales différences [24] [25] dans leurs publications concernant l'architecture de Dis. Ces points clés concernent principalement :

- Des instructions machine qui sont plus proches des processeurs actuels.
- Un mécanisme de ramasse miette plus efficace.
- etc.

3.1.7 Disco

Disco [26] est le résultat d'un projet de recherche de l'université de `Standford` qui avait pour but de rendre plus simple l'extension des systèmes d'exploitation modernes sur des systèmes possédant énormément de mémoire partagée, énormément de processeurs de type et de fonctions différentes avec le minimum possible d'effort d'implémentation.

Un point intéressant avec Disco est qu'il est l'invention de Edouard Bugnion [27] un Suisse qui est aussi l'un des fondateurs de la société `VMWare` (dont il fut le CTO et l'architecte jusqu'en 2004). Lors d'une conférence, Robert Grimm (professeur à l'université de New York) a comparé Disco à Xen [28].

Globalement, Disco est un hyperviseur basé sur un système multi-threadé pouvant fonctionner avec une grande quantité de mémoire. De cette manière, ce système permet de faire fonctionner un grand nombre de machines sur un même hardware. Disco virtualise toutes les ressource sur lesquels les systèmes virtuels doivent fonctionner. Le matériel exposé est un processeur de type MIPS dont la mémoire est contiguë et commence à 0 (modèle idéal). Les processeurs virtuels sont directement exécuté en parallèle des processeurs réel (ce qui assure les bonnes performances de la solution). Les allocations mémoire sont re-allouée par du logiciel se trouvant dans l'hypervieur à l'aide d'une simple table de transition (Adresse virutel <-> Adresse réel). Dernier point important, Disco propose des périphériques paravirutalisés (demandant des drviers particuliers) tel qu'un UART, un contrôleur SCSI, une carte Ethernet, etc.

3.1.8 Ensim

Ensim [29] est une solution de virtualisation et une entreprise. Aujourd'hui, `Ensim` est beaucoup plus actif dans le domaine de la supervision que de la virtualisation mais, pour ce chapitre, c'est leur solution de virtualisation qui est intéressante. Dans l'histoire de la virtualisation `Ensim` fait partie des pionniers.

Leur solution Ensim's Virtual Private Server (VPS) permet de partitionner un système d'exploitation et de le transformer en hébergeur d'"Appliance" isolées les unes des autres. Ensim permet un contrôle de la qualité de service, de l'isolation et de la supervision de ces services virtuels. Cette solution est disponible pour Solaris, Linux etWindows.

22. Article Wikipédia à propos de Dis : http://en.wikipedia.org/wiki/Dis_virtual_machine
23. Article Wikipédia à propos de inferno : http://en.wikipedia.org/wiki/Inferno_(operating_system)
24. Dis Virtual Machine Specification : http://doc.cat-v.org/inferno/4th_edition/dis_VM_specification
25. The design of the Inferno virtual machine : http://doc.cat-v.org/inferno/4th_edition/dis_VM_design
26. Publication décrivant Disco par son créateur : https://www.stanford.edu/class/cs240/readings/disco.pdf
27. Article Wikipédia à propos de Edouard Bugnion : http://en.wikipedia.org/wiki/Edouard_Bugnion
28. Comparaison de XEN et de Disco : https://cs.nyu.edu/rgrimm/teaching/sp11-os/vmms.pdf
29. Site officie de `Ensim` : http://www.ensim.com/

3.1.9 FreeBAS Jail

Les Jails[30] [31](prisons) de FreeBSD[32] sont un mécanisme qui permet de créer des environnements d'exécution isolés. Les jail utilisent chroot et par conséquent, chaque jail possède sa propre racine, mais les Jail ne s'arrête pas là. Les jail permettent aussi une restriction de visibilité sur les fichiers, les processus, les services réseaux, les autres Jail et peuvent aussi être restreinte à une seul adresse IP.

Les Jail sont implémentées dans FreeBSD grâce à des éléments du noyau qui sont développés pour être compatibles avec le mécanisme de jail.

3.1.10 Hive

Hive[33] est un système distribué qui a pour but l'utilisation simultanée de plusieurs noyaux ou cellules (terminologie hive). L'idée sous-jacescente à ce projet et d'augmenter là résistance d'un système en contenant les erreurs à seule la cellule infectée. Uniquement les processus contenus dans une cellules sont donc possiblement affectés. Afin de protéger aussi les processus internes à une cellule, chaque page mémoire et munie de permissions en écriture ce qui permet de cloisonner les processus entre-eux.

3.1.11 HP-UX Virtual Partitions

Les VPAR's (HP Virtual Partitions)[34] proposent une isolation des systèmes d'exploitation et des applications. Chaque VPAR exécute sa propre copie de HP-UX[35] (à l'image de IBM et de ces clusters de virtualisation). L'hyperviseur nommé vPar's permet de partager et d'assigner des ressources matérielles à des instances spécifiques. Comme la plupart des solutions de virtualisation modernes, VPAR permet la création dynamique de machines virtuelles.

3.1.12 Linux/RK

Linux/RK[36] est un système construit sur une base Linux. L'effort pour ce projet a été de construire une intégration de la qualité de service directement dans le noyau de Linux. Cette implémentation permet de tenir compte entre autre de l'implémentation physique du CPU, de la mémoire, du réseaux, etc.

3.1.13 LPAR

LPAR[37] est une technologie de partitionnement dynamique. LPAR permet de faire fonctionner plusieurs systèmes d'exploitation basés sur une image semblable sur un seul serveur. Cette technologie a été crée par IBM pour ses propres clusters.

Aujourd'hui, LAPR est utilisé dans la plupart des clusters IBM actuels.

3.1.14 Mac-on-Linux

Mac-on-Linux (MOL)[38] est une machine virtuelle dont l'implémentation permet d'émuler un processeur de type PowerPC et de faire fonctionner des systèmes Mac OS(7.5 à 9.2), Mac OS X et Linux. La plupart de l'implémentation de MOL est réalisée directement dans un module kernel (à l'image de KVM présenté plus loin dans ce document).

30. Article Wikipédia à propos des jails FreeBSD : http://en.wikipedia.org/wiki/FreeBSD_jail
31. Page développement des Jail FreeBSD : http://en.wikipedia.org/wiki/FreeBSD_jail
32. Site officiel de FreeBSD : http://www.freebsd.org/fr/
33. HIVE: OPERATING SYSTEM FAULT CONTAINMENT FOR SHARED-MEMORY MULTIPROCESSORS : ftp://reports.stanford.edu/pub/cstr/reports/csl/tr/97/712/CSL-TR-97-712.pdf
34. Introducing HP-UX 11i Virtual Partitions : http://h20338.www2.hp.com/hpux11i/downloads/vPars_Intro_WP_26Aug05_final.pdf
35. Article Wikipédia à propos de HP-UX : http://fr.wikipedia.org/wiki/HP-UX
36. Site officiel du projet Linux/RK : https://www.cs.cmu.edu/~rtmach/
37. Article Wikipédia à propos de LAPR : http://en.wikipedia.org/wiki/LPAR
38. Article Wikipédia à propos de MOL : http://en.wikipedia.org/wiki/Mac-on-Linux

3.1.15 MAE

Macintosh Application Environnment (MAE) [39] est une application fonctionnant sur des processeurs de type RISC (Comme par exemple les SPARC de sun) et permet de virtualiser un environnement Macintosh. L'environnement virtualisé propose un processeur virtuel de type Motorola 68LC040.

3.1.16 Microsoft Virtual Server

Microsoft [40] a dans le passé proposé sa propre solution de virtualisation. Par exemple, Windows NT proposait des sous-systèmes virtualisés tel que des machines DOS virtualisées (VDM), des machines Windows (WOW [41]), des machines Windows 16-bit, des composants OS/2, un sous-système POSIX et des fonctionnalités Win32. OS/2 et Win32 fonctionnaient comme des processus à par entière de type serveur tendit que Win16 et DOS fonctionnaient eux dans le contexte de machines virtuelles. Tous ces éléments étaient dépendants du système natif NT de Windows.

VDM [42] était dérivée du code de MS-DOS 5.0 et proposait un processeur x86 virtualisé basique. Cet émulateur basique proposait aussi un intercepteur de "trap" afin d'exécuter les instructions privilégiées. De manière similaire, Windows 95 utilisait des machines virtuelles pour faire fonctionner de vielles applications (windows 3.x et DOS).

Microsoft a inclus beaucoup plus de fonctionnalités de virtualisation dans ses serveurs depuis son acquisition de Connectix en 2003. Ce rachat a permis la création de la solution de virtualisation Virtual PC. Aujourd'hui, comme beaucoup d'entreprise, Microsoft essaie de rendre ces applications de plus en plus virtualisées comme par exemple ce fut le cas avec Microsoft SQL server 2000 qui permettait plusieurs instances d'un même serveur.

3.1.17 Nemesis

Nemesis [43] [44] est un système créé à l'université de Cambridge pour réaliser un système dédié à la qualité de service. Le noyau de Nemesis est extrêmement épuré, la majorité du code étant exécuté directement dans les processus applicatifs eux-mêmes.

Nemesis possède un noyau capable de gérer à très bas niveau la gestion de l'affectation des tâches aux différentes parties des processeurs présents. Le traitement de la mémoire est lui aussi particulier. Nemesis englobe en effet toutes les mémoires disponibles dans une seule et grande table. Ensuite, des éléments de cette table sont spécifiquement alloués à chaque processus. Comme le noyau n'effectue qu'un minimum de travail, Nemesis permet donc d'éviter au maximum les effets d'un processus sur les autres processus (Situation souvent d'écrite comme du "QoS crosstalk" [45]).

Les travaux effectués sur Nemesis sont actuellement utilisés dans plusieurs systèmes tels que Pebble [46], V++ Cache Kernel [47], etc.

3.1.18 Plex86

Plex86 [48] propose une implémentation légère de vritualisation basée x86 sur Linux. La particularité de ce système est d'être purement exécutif (il ne supporte aucune instruction qui ne peut pas être virtualisées) et ne propose aucune I/O virtualisées.

39. Article Wikipédia à propos de MAE : http://en.wikipedia.org/wiki/Macintosh_Application_Environment
40. Article Wikipédia à propos de Microsoft Virtual Server : http://en.wikipedia.org/wiki/Microsoft_Virtual_Server
41. Article Wikipédia à propos de WOW : http://en.wikipedia.org/wiki/Windows_on_Windows
42. Article Wikipédia à propos de VDM : http://en.wikipedia.org/wiki/Virtual_DOS_machine
43. Nemesis Kernel Overview : http://www.cl.cam.ac.uk/research/srg/netos/old-projects/pegasus/publications/overview/brief-overview.html
44. Nemesis publications : http://www.cl.cam.ac.uk/research/srg/netos/old-projects/nemesis/documentation.html
45. Article Wikiépdia à propos de la qualité de service : http://en.wikipedia.org/wiki/Quality_of_service
46. Publication du BELL labs à propos de pebble : http://www.usenix.org/event/usenix99/full_papers/gabber/gabber.pdf
47. A Caching Model of Operating System Kernel Functionality : http://www.usenix.org/publications/library/proceedings/osdi/full_papers/cheriton.a
48. Page officiel de Plex86 : http://plex86.sourceforge.net/

Actuellement, de mineures modifications sur un noyau linux permettent de le rendre exécutable dans Plex86.

Tout comme pour Bochs, Plex86 est un logiciel qu'il est intéressant de démonter pour l'étude et la compréhension d'un logiciel de virtualisation.

3.1.19 Progamming Language Virtual Machines

Certains langages de programmation sont implémentés en utilisant une machine virtuelle pour leurs exécutions. L'exécution dans un environnement virtualisé apporte des avantages évidents tels que l'isolation et la portabilité.

Dans les années 70, le système UCSD P-System [49] utilisait une machine virtuelle qui exécutait des instructions en "p-code" (akin to bytecode). Le langage utilisé était du (UCSD PASCAL) langage très populaire à l'époque.

Le point suivant s'intéresse à la machine vrituelle de Java actuellement très utilisée et qui repose sur les même principes. Bien que évidement d'autre langages utilisent ce principe.

La JVM de Java

la Java Virtual Machine (JVM) [50] [51] est actuellement l'une des machines virtuelle les plus connues. De manière générale, la JVM est un ordinateur complet. Ces spécifications décrivent de manière très précises la façon dont chaque élément doit fonctionner (utilisation des registres, de la pile, du tas, du ramasse miettes, du pointeur d'instructions, etc.). Une implémentation de JVM est donc une plateforme à part entière dédiée à un système particulier comme par exemple X86/Linux, x86/windows, SPARC/Solaris, etc.

Point intéressant, il est possible de placer une JVM dans du microcode [52] [53] ou même directement sur du hardware (Picojava [54] est une implémentation hardware d'une JVM).

Comme pour tous les langages basés sur une machine virutelle, il est possible de compiler du code java sur n'importe quelle architecture et de l'exécuter sur n'importe quelle autre (à condition qu'elles dispose de toutes les dépendances nécessaires à l'exécution du code).

Il est intéressant de comprendre qu'une machine virtuelle Java exécute du bytecode et pas du java, il est donc possible d'exécuter du code provenant de n'importe quel langage pourvu qu'il aie été compilé en bytecode compatible avec les JVM

3.1.20 QLinux

QLinux [55] [56] est un système d'exploitation dérivé de Linux et dédié à la qualité de service. Il est le résultat de recherches menées par le MIT et l'université du Texas. QLinux inclus une notion de répartition de charges hiérarchiques tant pour les tâches que pour pour les communications réseaux, etc.

3.1.21 Simics

Simics [57] est une plateforme complète de simulation. Simics a été développé en essayant de maintenir une certaine balance entre les performances et la tolérance aux pannes. Actuellement Simics est disponible pour de nombreuses platformes (X86/Linux, X86/Windows, Sparc/solaris) et permet d'en émuler de nombreuses aussi (RM, MIPS, PowerPC, SPARC, x86, AMD64, IA64).

49. Article Wikipédia à propos de UCSD P : http://en.wikipedia.org/wiki/UCSD_Pascal
50. Article Wikipédia à propos de la machine virtuel Java : http://en.wikipedia.org/wiki/Java_Virtual_Machine
51. The Java Virtual Machine Specification : http://java.sun.com/docs/books/jvms/
52. Article Wikipédia à propos du microcode : http://en.wikipedia.org/wiki/Microcode
53. JOP Design Flow : https://www2.imm.dtu.dk/~masca/slides/JOPBuild.pdf
54. Article Wikipédia à propos de picojava : http://en.wikipedia.org/wiki/PicoJava
55. Page officiel de QLinux : http://lass.cs.umass.edu/software/qlinux/
56. Application Performance in the QLinux Multimedia Operating System : ftp://ftp.cs.umass.edu/pub/techrept/techreport/2004/UM-CS-2004-013.ps
57. Article Wikipédia à propos de Simics : http://en.wikipedia.org/wiki/Simics

3.1.22 SimOS

SimOS [58] [59] est un simulateur développé à l'université de Standford. SimOS est capable de modéliser un ordinateur virtuel complet (CPU, caches,bus multiprocesseurs, périphériques réseaux, disques, etc.) et permet aussi un contrôle fin du niveau de simulation de chaqu'un de ces éléments. Son architecture et son modèle sont très proches de ceux utilisés par Simics.

Le développement de SimOS à commencé en 1993 avec pour but la simulation de matériel pour une architecture SPARC. Les versions suivantes ont ensuite permis de simuler du MIPS (version SGI avec support de IRIX). Le développement de la version MIPS à permis de developper les premiers principed d'exécution directe (sur le hardware) d'une machine de type MIPS de SGI. Ensuite, SimOS à étendu sont support au processeurs Alpha et PowerPC.

3.1.23 Solaris

SUN à inclu le partitionnement statique des ressources serveur en 1996 avec les ordinateurs de la famille E10K [60]. Les partitions ou "domaine" étaient définies en allouant à chaque instance une part précise de matériel réel. A l'époque, chaque domaine exécutait sa propre copie de Solaris. En 1999 SUN introduisit le partitionnement dynamique qui permettait la migration de domaines d'une machine à une autre.

En 2002 SUN introduisit le concept de "Solaris Containers" qui étaient des environnement d'exécution limité en ressources qui permettaient d'exécuter une copie de Solaris. Au fil du temps, SUN a ajouté de plus en plus de fonctionnalités à ces conteneurs pour les faire devenir des zone avec l'arrivée de Solaris 10.

Le concept de "zone Solaris" est un principe dérivé du concept des jail BSD et était aussi appelé "trusted container" [61]. Une zone apparait pour le système comme une machine réelle même si en réalité, il ne s'agit que 'une copie du noyau de Solaris.

3.1.24 User Mode Linux

User Mode Linux ou UML [62] est un portage du noyau Linux vers l'architecture abstraite "um". En fait, il s'agit d'un portage du noyau Linux lui donnant la capacité d'être exécuté par lui-même. Les machines virtuelles UML fonctionnent donc sur le système hyperviseur comme de simple processus.

UML a originellement été développé pour créer un mode dis "trace thread" dans le noyau Linux. Ce mode permettait à des thread particuliers appartenant à des processus UML d'être capables de communiquer entre le système virtualisé et non virtualisé.

Il existe des produits dérivés de UML comme par exemple UMLinux [63] permettant de faire de la simulation réseau.

3.1.25 D'autres exemples

Il existe bien sûr un très grand nombre d'autres solutions liées au domaine de la virutalisation comme par exemple :

- Les solutions VMWare.
- z/VM de IBM.

58. Article Wikipédia à propos de SimOS : http://en.wikipedia.org/wiki/SimOS
59. Linux/SimOS - A Simulation Environment for Evaluating High-speed Communication Systems : http://academic.csuohio.edu/yuc/papers/Ben-ICPP2002.pdf
60. Site regroupant des informations sur la série E10K : http://www.cray-cyber.org/systems/e10k.php
61. Article Wikipédia à propos des conteneur Solaris : http://en.wikipedia.org/wiki/Solaris_Containers
62. Site officiel de UML : http://user-mode-linux.sourceforge.net/
63. Site offiicel de UMLinux : http://www3.informatik.uni-erlangen.de/Research/UMLinux/linuxtag2002/linuxtag2002_all.html

- `Cellular IRIX`[64].
- XEN[65]
- etc.

3.2 Conclusion

Ce chapitre s permis de voir un petit nombre de solutions de virtualisation parmi la très grande diversité disponible. Néanmoins il permet de se rendre compte du vaste domaine qu'est l'émulation et la virtualisation et ainsi de se rendre compte qu'il existe un grand nombre d'approche pour résoudre un même type de problème.

Le chapitre suivant commence l'étude de la solution de virtualisation KVM reposant sur beaucoup de principes vu ici mais en apportant aussi de nouveaux.

64. Documentation à propos de Cellular IRIC : http://www.sgistuff.net/software/irixintro/documents/irix6.4TR.html
65. Site officiel de XEN : http://www.xen.org/

II

KVM (Kenerl-base Virtual Machine)
La virtualisation des années 2000

Sommaire

Chapitre

4

Architecture de KVM (Kernel-base Virtual Machine)

A dreamer is one who can only find his way by moonlight, and his punishment is that he sees the dawn before the rest of the world

Oscar Wilde.

Kernel-based Virtual Machine (KVM) est sans doute l'une des solutions de virtualisation la plus moderne par la manière dont elle a été imaginée et par les principes mis en œuvre. L'objectif de ce projet a été depuis le début de créer un hyperviseur moderne et performant reposant sur les innovations réalisées ces dernières années dans le monde de Linux et des systèmes d'exploitation.

Les chapitres précédents ont présenté les buts globaux de ce projet. Pour comprendre les avantages de KVM ainsi que ces forces et faiblesses il est utile de comprendre comment cet hyperviseur fonctionne et comment il est construit. L'étude d'état de l'art de ce document à essayer avec l'aide d'un historique et un tour d'horizon de familiariser le lecteur avec les grandes technologies de la virtualisation. Il est maintenant possible de commencer l'étude plus en profondeur de la solution de virtualisation KVM.

Les chapitres qui composent cette partie du document passeront en revue les différents points choisis pour expliquer ce qu'est KVM. Ce premier chapitre présentera l'architecture globale de KVM puis entrera dans l'hyperviseur de manière plus détaillée. A la demande des mandants de ce projet, quand il est possible et justifié de le faire, des pointeurs sur le code source de KVM seront présentés. Ces pointeurs doivent permettre d'aider à comprendre certains choix effectués par les développeurs de KVM, mais aussi à aider le laboratoire de virtualisation de `HEPIA` à comprendre cette solution de virtualisation de manière plus précise. Certaines de ces indications seront donc parfois directement montrées en rapport à du code commenté et d'autres fois simplement comme indication où se trouve l'information complète.

La compréhension de certaines décisions d'implémentation (par le manque de documentation complète sur KVM) demande encore beaucoup de lecture de code pour les comprendre. Pour ce document, il a été choisi de présenter les plus importantes pour tenir dans les délais imposés.

Pour les personnes intéressées par une introduction moins technique et plus orientée opérationnelle de KVM, l'auteur de ce document recommande très fortement le document suivant [1].

1. Présentation de KVM par `KVM` : `https://www.redhat.com/f/pdf/rhev/DOC-KVM.pdf`

4.1 Architecture global

De manière générale, l'idée fondamentale derrière KVM est de pouvoir (à terme) transformer n'importe quel Linux en hyperviseur simplement en chargeant un module du noyau. En effet, KVM est construit sous la forme d'un "Loadable Kernel Module" qui lui permet d'être chargé à n'importe quel moment dans la vie d'un système.

`Loadable Kernel Module (LKM) ou module de noyau chargeable :`

Dans un système d'exploitation, un module est une partie du noyau qui peut être intégrée pendant son fonctionnement. Le terme anglais généralement employé pour les désigner est Loadable Kernel Module, abrégé LKM, ou en français : « module de noyau chargeable ». Cette fonctionnalité existe dans les noyaux Linux et les noyaux BSD.

C'est une alternative aux fonctionnalités compilées dans le noyau, qui ne peuvent être modifiées qu'en relançant le système. [2] [3]

Ce module crée un périphérique dans l'espace utilisateur du système. Ce périphérique s'appelle /dev/kvm et représente l'hyperviseur KVM tel qu'entendu dans la définition habituelle d'un hypervieur. En général, les outils permettant de créer et utiliser des machines virtuelles KVM interagiront avec ce périphérique particulier. Ce périphérique permet aussi au noyau de fonctionner dans un troisième mode (en complément des modes système et utilisateur) il s'agit du mode "guest" (mode dans lequel les machines virtuelles sont exécutées) qui sera expliqué plus en détail dans la suite de ce chapitre. A l'aide de ce périphérique et de ce mode de fonctionnement particulier, chaque machine virtuelle possède son propre espace d'adressage qui sera différent de celui du noyau, mais résident dans l'espace utilisateur du système. Ce mécanisme fait donc intégralement partie de la virtualisation et des mécanismes d'isolation mis en œuvre.

KVM permet donc très simplement de transformer un noyau Linux en hyperviseur (A l'aide de l'installation et l'activation du module). De part son intégration directe dans le noyau Linux, KVM en temps que solution de virtualisation bénéficie de nombreux services déjà fournis par le noyau de Linux ainsi que de futurs services qui seront développés dans l'avenir. Par exemple :

- Support matériel étendu.
- Mécanisme d'allocation mémoire éprouvé.
- Scheduler récemment refait (CFS [4])
- Sécurité du système et utilitaires de sécurité disponibles.
- Etc.

Mais, KVM n'est pas le premier système a avoir essayé de transformer Linux en un hyperviseur. User mode Linux (UML) [5] réalise ceci en permettant de virtualiser des serveurs Linux ou Windows de manière simple mais, sans pour autant atteindre les performances nécessaires à des environnements de production avancé.

Pour réutiliser de manière optimale tous les mécanismes déjà présents dans Linux avec KVM, chaque machine virtuelle est contenue dans un processus Linux distinct. Cette approche permet une réutilisabilité maximale des services disponibles.

La figure suivante 4.1 présente un système Linux depuis le hardware jusqu'aux processus applicatifs. La couche basse (en gris) représente le matériel, la couche suivante (en vert) présente le noyau Linux qui contient les drivers qui permettent d'utiliser le hardware. Le noyau se charge aussi de l'allocation de la mémoire, de la répartition des processeurs auprès des processus, ... Les couches hautes du graphique représente l'espace utilisateur avec les processus applicatifs (orange) et les machines virtuelles contenues dans des processus (conteneurs) distincts (en bleu). Cette illustration présente aussi le fait que les processus virtualisés doivent passer par

2. Article Wikipédia sur les LKM : http://en.wikipedia.org/wiki/Loadable_kernel_module
3. Article avancé "Anatomy of Linux loadable kernel modules" de IBM : http://www.ibm.com/developerworks/linux/library/l-lkm/
4. Article de IBM à propos du nouveau scheduler : http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/
5. Site officiel de UML : http://user-mode-linux.sourceforge.net/

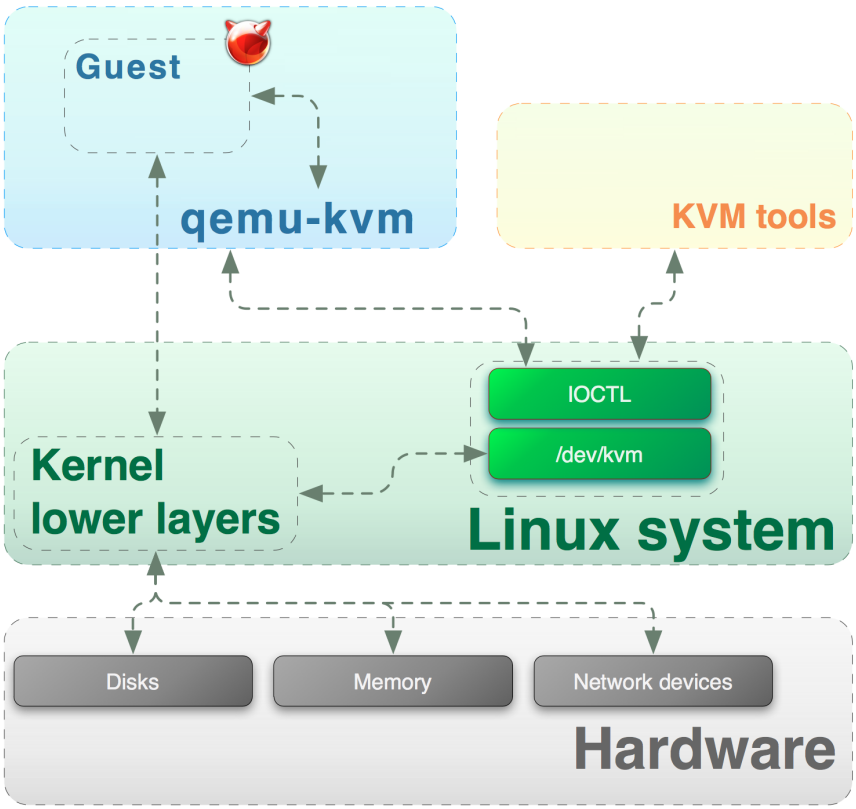


FIGURE 4.1: Architecture global de KVM

l'intermédiaire des services fournis au travers du device /dev/kvm pour faire fonctionner la machine virtuelle qu'il contienne. Cette affirmation n'est pas complètement claire mais, de prochains chapitres expliqueront dans quels cas d'utilisation /dev/kvm est indispensable.

Cette section a présenté de manière globale comment était construit un système KVM. Les sections suivantes vont maintenant présenter plus en détail des parties choisies de cette architecture.

4.1.1 Architecture de l'hyperviseur

Globalement, KVM est composé de quatre parties (4.2). (1), le système de virtualisation situé dans le noyau de Linux (2). Tel que discuté précédemment, il s'agit de tout ce qui se trouve derrière le périphérique /dev/kvm et qui est chargé de la création, maintenance, etc. des machines virtuelles dans leurs espaces d'exécution respectifs.

Dans l'espace utilisateur, mais ne pouvant interagir que dans un espace qui leur est propre se trouvent les machines virtuelles (5) intégrées dans leurs conteneurs (4). Dans KVM, du côté utilisateur, un conteneur de machine virtuelle apparait comme un simple processus tel que pourrait l'être Firefox, Grep, etc. Du point de vue interne de ces processus, leur visibilité du système est réduite (par leurs présences en espace utilisateur) et leurs capacités par rapport à des processus habituels sont augmentées de fonctionnalités liées à leurs nouvelles utilités. Ces processus sont donc des processus normaux du point de vue système, mais d'un point de vue architecture, il faut considérer un processus en espace utilisateur utilisé par KVM comme un conteneur tandis que le système virtualisé est le contenant.

La troisième partie sont les utilitaires (3) en espace utilisateur capables d'interagir avec le périphérique /dev/kvm. Il est en effet très compliqué d'utiliser sans utilitaire l'interface /usr/bin/qemu-kvm à la main (l'interface /usr/bin/qemu-kvm est expliquée plus tard dans ce document, mais, globalement, il s'agit du conteneur de machine virtuelle cité précédemment).

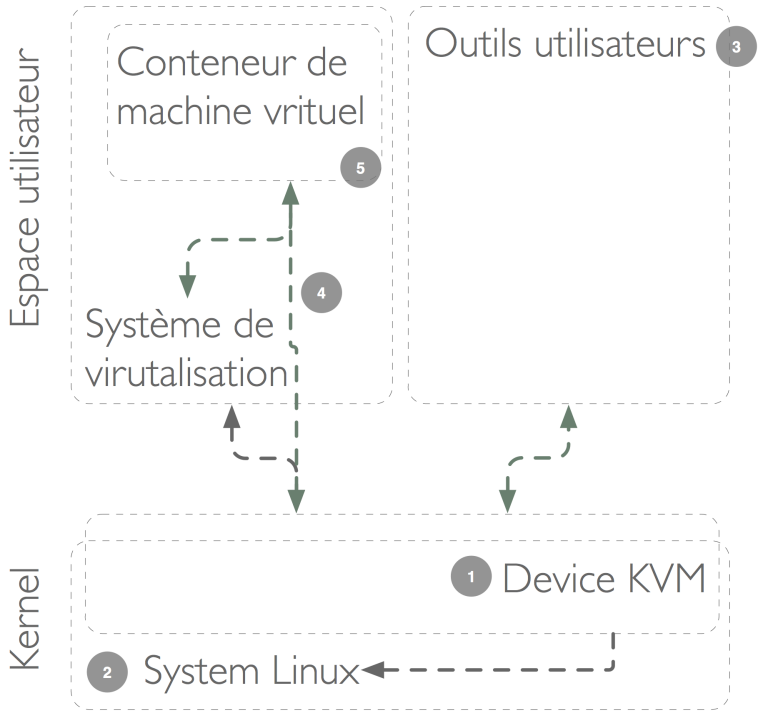


FIGURE 4.2: Découpage de la solution de virtualisation KVM

Par exemple, la capture suivante montre à titre informatif les étapes nécessaires pour démarrer à la main (créer un conteneur, le configurer, démarrer le système virtualisé, etc) un conteneur pour une machine virtuelle.

```
/usr/bin/qemu-kvm -S -M pc-0.13 -enable-kvm -m 512 -smp 2,sockets=2,cores=1,threads=1
-name test -uuid e9b4c7be-d60a-c16e-92c3-166421b4daca -nodefconfig -nodefaults
-chardev socket,id=monitor,path=/var/lib/libvirt/qemu/test.monitor,server,nowait
-mon chardev=monitor,mode=readline -rtc base=utc -boot c
-drive file=/var/lib/libvirt/images/test.img,if=none,id=drive-virtio-disk0,boot=on,format=raw
-device virtio-blk-pci,bus=pci.0,addr=0x5,drive=drive-virtio-disk0,id=virtio-disk0
-drive if=none,media=cdrom,id=drive-ide0-1-0,readonly=on,format=raw
-device ide-drive,bus=ide.1,unit=0,drive=drive-ide0-1-0,id=ide0-1-0
-device virtio-net-pci,vlan=0,id=net0,mac=52:54:00:cc:1c:10,bus=pci.0,addr=0x3
-net tap,fd=59,vlan=0,name=hostnet0 -chardev pty,id=serial0 -device isa-serial,chardev=serial0
-usb -device usb-tablet,id=input0 -vnc 127.0.0.1:0 -vga cirrus -device AC97,id=sound0,bus=pci.0,addr=0x4
-device virtio-balloon-pci,id=balloon0,bus=pci.0,addr=0x6
```

Bien que compréhensible à la lecture, il est facile de se rendre compte de l'utilité d'outils nous simplifiant le travail. De part leurs caractères indispensables, ces outils sont considérés comme la troisième partie de KVM. Les outils fournis par défaut seront expliqués ainsi qu'une librairie aujourd'hui indispensable pour utiliser la plupart de utilitaires aidant à utiliser KVM. Ces chapitres s'intéresseront à leurs architectures plus qu'à leurs utilisations. Une des parties principales de ce document (pratique de KVM) se charge de traiter de l'aspect pratique de l'utilisation de KVM.

4.2 Moteur de KVM

Tel qu'il a été discuté précédemment, KVM est à la base le périphérique /dev/kvm. Pour être plus précis, KVM est en fait un driver pour utiliser ce périphérique. Ce périphérique est donc une façade qui expose les

fonctionnalités servant à créer des machines virtuelles qui ont leur propre espace mémoire, mais qui font encore partie de l'espace utilisateur de Linux.

Le périphérique /dev/kvm est construit comme un périphérique orienté caractère[6]. Ce périphérique comme la plupart des périphériques peut être contrôlé et utilisé à partir de l'espace utilisateur au travers des ioctl's[7][8] dédiés. Ces ioctl's sont traités et maîtrisés par l'interface "générique" de ce drivers. Le code principal de cette interface se trouve dans Linux/virt/kvm/kvm main.c (le code du driver KVM se trouve dans : Linux/virt/kvm). Les implémentations par architecture sont elles disponibles dans linux/arch/x86/kvm. Prenons le cas des ioctl précédemment citées. La liste complète des isoctl's pour la manipulation de /dev/kvm est disponible dans la documentation du noyau (/linuxDocumentation/kvm/api.txt), leur code source est disponible dans l'implémentation du driver KVM (linux/arch/x86/kvm/x86.c pour leur implémentation sous x86). Comme tous les événements pouvant affecter ce type de périphérique, leur définition sont dépendante d'un fichier ".h" qui se trouve dans linux/include/linux/kvm.h.

Ces contrôles ont pour fonction de piloter le périphérique /dev/kvm. Ce périphérique propose toutes les fonctionnalités nécessaires à la virtualisation dont par exemple :

- Créer une nouvelle machine virtuelle (on parle de la machine en elle même et pas de l'image du système qu'elle animera).
- Allouer de la mémoire à une machine virtuelle.
- Écrire et lire des registres dans les processeurs virtuels.
- Injecter des interruptions dans un processeur virtuel.
- Démarrer un processeur virtuel.
- Créer des APIC.
- Etc.

De manière générale, le flux de contrôle du device ce fait donc comme la figure 4.1 le présente. Un logiciel qui a besoin de communiquer avec le conteneur d'une machine virtuelle ou avec le périphérique KVM va utiliser les fonctionnalités mises à disposition par les ioctl's au travers de linux/include/linux/kvm.h. Ces fonctionnalités, en fonction de l'architecture pour laquelle le noyau est compilé vont aller chercher le code correspondant dans linux/arch/XX/kvm.

L'exemple suivant volontairement simplifié présente globalement le démarrage d'une machine virtuelle avec les fonctions proposées par les ioctl's. Pour rendre le tout plus simple, les méthodes utilisées sont des méthodes disponibles avec les outils de l'espace utilisateur de KVM (kvm-qemu/qemu-kvm/qemu-kvm.c).

```
#include <linux/kvm.h>

/*
Debut du programme conteneur
*/
int main()
{
void *vm_mem; //mémoire utilisée pour le conteneur
int fd; //descripteur de fichier pointant sur le device KVM
int vm_id; //identification de la machine virtuelle
/*
Initialisation du device KVM
*/
```

6. Linux Kernel Development : Chapitre 17
7. Article Wikipédia à propos des ioctl's : http://en.wikipedia.org/wiki/Ioctl
8. Livre Linux Kernel Development

```
fd = open("/dev/kvm", O_RDWR);
    if (fd == -1) {
        perror("open /dev/kvm");
        return -1;
    }

    /*
     Toutes les étapes de création d'un contexte ne sont pas montrées
    */
.
    kvm_state = qemu_mallocz(sizeof(*kvm_state));
    kvm_context = &kvm_state->kvm_context;
    kvm_state->fd = fd;
    .

    /*
     Création du contexte (description system de la machine)
    */
    kvm = kvm_create_context();

close(fd);

/*
Création d'une machine virtuel avec 128*1024*1024 (128MB) de mémoire
*/
    r = kvm_create_vm(kvm); //création du conteneur selon le contexte
    if (r < 0) {
        return r;
    }
    /*
      Création de la partie hardware de la machine
    */
    r = kvm_arch_create(kvm, 128*1024*1024, vm_mem);
    if (r < 0) {
        return r;
    }
    /*
      Création de l'espace physique visible par la VM.
    */
    r = kvm_create_default_phys_mem(kvm, 128*1024*1024, vm_mem);
    if (r < 0) {
        return r;
    }
    /*
     Création du contrôleur d'interruption nécessaire pour être capable de les gérer
    */
    kvm_create_irqchip(kvm);


//On charge le fichier arg[2] (second argument imaginaire) dans
//notre mémoire à partir de l'adresse 0x100000 (pour rappel, la pile grossis vers le bas)
load_file(vm_mem + 0x100000, arg[2]);

//on démarre la machine
kvm_run(kvm, 0);
```

```
return 0;
}
```

Avec l'aide des fichiers sources et de /linuxDocumentation/kvm/api.txt il est possible de comprendre ce conteneur éducatif (ce code présente un conteneur simplifié). La première partie du code se charge d'ouvrir le périphérique /dev/kvm , crée une machine virtuelle et fournit en retour un pointeur sur la description de la machine. Dans la terminologie utilisée dans le code de KVM cette description est appelée contexte ou statut KVM.

La seconde partie du code initialise le hardware(bas niveau : CPU, APIC, etc.) virtuel de la machine puis, définit l'espace mémoire utilisable par celle-ci. Puis, à l'aide de la fonction imaginaire load file, le fichier qui contient l'image de la machine à virtualiser est déposé dans la mémoire du conteneur.

Le code ci-dessous présente la fonction kvm create vm et permet de voir le déclenchement de l'ioctl KVM CREATE VM.

```
int kvm_create_vm(kvm_context_t kvm)
{
 .
 .
    fd = kvm_ioctl(kvm_state, KVM_CREATE_VM, 0);
    if (fd < 0) {
        fprintf(stderr, "kvm_create_vm: %m\n");
        return -1;
    }
    kvm_state->vmfd = fd;
    return 0;
}
```

L'ioctl utilisé ici à la fonction suivante :

```
4.2 KVM_CREATE_VM

Capability: basic
Architectures: all
Type: system ioctl
Parameters: none
Returns: a VM fd that can be used to control the new virtual machine.

The new VM has no virtual cpus and no memory.  An mmap() of a VM fd
will access the virtual machine's physical address space; offset zero
corresponds to guest physical address zero.  Use of mmap() on a VM fd
is discouraged if userspace memory allocation (KVM_CAP_USER_MEMORY) is
available.
```

La dernière fonction du code KVM RUN déclenche le démarrage de la machine.

Ce court exemple montre qu'il est simple (du point de vue code) d'interagir avec le périphérique fourni pour la virtualisation et permet de comprendre ce qui se trouve derrière les outils d'aide à la virtualisation. Actuellement, dans la plupart des cas, les outils ne font plus appel directement à des ioctl, mais se base sur des librairies ou des fonctionnalités qui y font appel. Comme par exemple celles utilisées dans notre exemple qui font partie des applicatifs en espace utilisateur de KVM.

Le sous chapitre suivant présente de manière informative les différents ioctl nécessaires pour créer et démarrer une machine virtuelle.

- KVM CREATE VM : Crée de manière générique la future machine virtuelle, mais sans matériel.
- KVM SET USER MEMORY REGION : Alloue l'espace utilisateur que peut adresser la machine.

- KVM CREATE IRQCHIP / ...PIT KVM CREATE VCPU : Crée les différents composants hardware.
- KVM SET REGS / ...SREGS / ...FPU / ... KVM SET CPUID / ...MSRS / ...VCPU EVENTS / ... KVM SET LAPIC Crée les configurations matérielles nécessaires au bon fonctionnement de la machine.
- KVM RUN : Démarre la machine virtuelle.

Dans l'exemple de code précédent, il y a donc des raccourcis qui font que la machine créée ne démarra jamais, mais comme le montre cette liste, il est difficile en quelques lignes de créer un exemple simple et court. Cependant, d'un point de vue purement technique, le code pour maintenir et créer une machine KVM est simple et claire.

4.2.1 Architecture du périphérique KVM

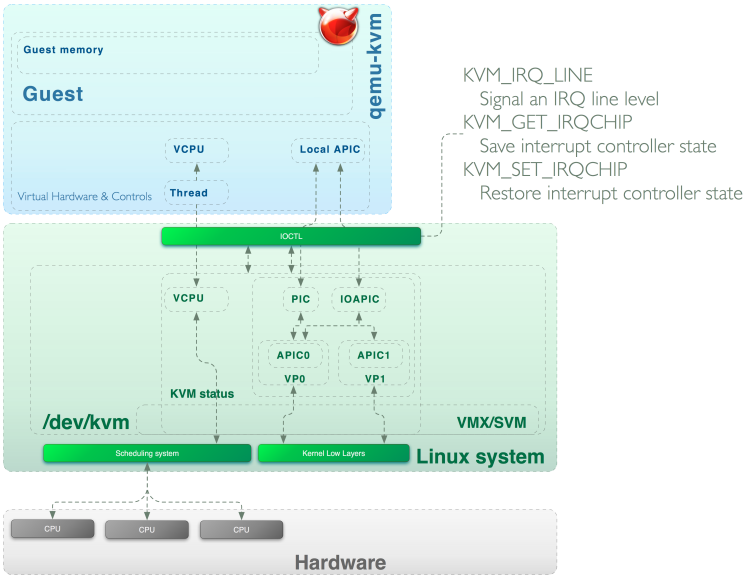


FIGURE 4.3: Architecture du périphérique KVM

L'illustration 4.3 présente comment est construit le périphérique /dev/kvm. Pour chaque machine virtuelle, il existe un contexte (statut KVM) qui contient la configuration hardware bas niveau (CPU, APIC, etc) utilisée. Cette configuration est créée et paramétrée à l'aide des ioctl. En dessous de ces statuts, afin de proposer de bonne performance, se trouve la couche (VMX/SVM) qui fait le lien avec les optimisations hardware disponibles dans les processeurs récents. Le reste de l'illustration est présent à titre informatif et sera expliqué plus tard dans ce document.

Le code utilisé par cette couche basse qui lie le hardware à KVM est disponible dans le fichier linux/arch/x86/kvm/vmx.c.

4.3 Outils utilisateurs de KVM

Pour que l'utilisateur n'aie pas à directement à utiliser les ioctl et l'accès au périphérique KVM, des outils ont été développé pour assurer la gestion de base des machines virtuelles. Parmi les outils disponibles, certains sont particulièrement importants pour ce travail.

4.3.1 qemu-img

qemu-img permet de créer une image qui contiendra une machine virtuelle. De très nombreux arguments sont disponibles pour créer une machine virtuelle, mais les plus importants sont décrits avec la capture suivante.

```
qemu-img create -f qcow file Size
```

- `create` : Commande la création d'une image.
- `-f qcow` : Format de la machine virtuelle. Le format présenté dans l'exemple est qcow[9].
- `file Size` : Taille brute de l'espace (disque) disponible pour la machine virtuelle.

par exemple, pour créé une image qcow de 10 Gigabyte :

```
qemu-img create –f qcow kvm/Fedora.img 10G
```

4.3.2 qemu-kvm

qemu-kvm est le conteneur qui contient le matériel virtualisé qui est exposé à la machine virtuelle ainsi que les outils de contrôle nécessaires. La commande suivante présente quelques arguments qu'il est possible d'utiliser pour contrôler le conteneur.

```
qemu-kvm disk –net nic,model=rtl8139
-net user -soundhw es1370 -m memory
-cdrom install-image  -boot d -daemonize
```

- `-net nic,model=rtl8139` : Modèle de carte réseau émulée
- `-net user` : Émulation réseau simple
- `-soundhw es1370` : Modèle de carte son émulée
- `-m memory` : La machine possède "memory" espace mémoire RAM disponible
- `-cdrom install-image` : Le CD-ROM est monté avec la machine et utilise l'image "install-image"
- `-boot d` : Démarre sur le CD-ROM
- `-daemonize` : Fonctionne en temps que daemon/Tâche de fond (FORK)

Il est cependant souvent nécessaire de sélectionner beaucoup plus d'arguments pour faire fonctionner une machine virtuelle configurée comme on le souhaite.

```
/usr/bin/qemu-kvm -S -M pc-0.13 -enable-kvm -m 512 -smp 2,sockets=2,cores=1,threads=1
-name test -uuid e9b4c7be-d60a-c16e-92c3-166421b4daca -nodefconfig -nodefaults
-chardev socket,id=monitor,path=/var/lib/libvirt/qemu/test.monitor,server,nowait
-mon chardev=monitor,mode=readline -rtc base=utc -boot c
-drive file=/var/lib/libvirt/images/test.img,if=none,id=drive-virtio-disk0,boot=on,format=raw
-device virtio-blk-pci,bus=pci.0,addr=0x5,drive=drive-virtio-disk0,id=virtio-disk0
-drive if=none,media=cdrom,id=drive-ide0-1-0,readonly=on,format=raw
-device ide-drive,bus=ide.1,unit=0,drive=drive-ide0-1-0,id=ide0-1-0
-device virtio-net-pci,vlan=0,id=net0,mac=52:54:00:cc:1c:10,bus=pci.0,addr=0x3
-net tap,fd=59,vlan=0,name=hostnet0 -chardev pty,id=serial0 -device isa-serial,chardev=serial0
-usb -device usb-tablet,id=input0 -vnc 127.0.0.1:0 -vga cirrus -device AC97,id=sound0,bus=pci.0,addr=0x4
-device virtio-balloon-pci,id=balloon0,bus=pci.0,addr=0x6
```

Il est fort heureusement aujourd'hui possible d'utiliser des outils qui nous facilite la tâche. Le chapitre à propos de libvirt et celui à propos de la pratique de KVM présentent ces différents outils.

4.4 Présentation d'un conteneur qemu-kvm

Ce chapitre propose d'observer de manière plus détaillée un conteneur de virtualisation qemu-kvm. Une partie du cloisonnement et de la virtualisation est traitée dans ces conteneurs. La figure suivante (4.4) illustre de manière simplifiée ce que l'on peut trouver dans un de ces conteneurs.

9. Article Wikipédia à propos du format qcow : http://en.wikipedia.org/wiki/Qcow

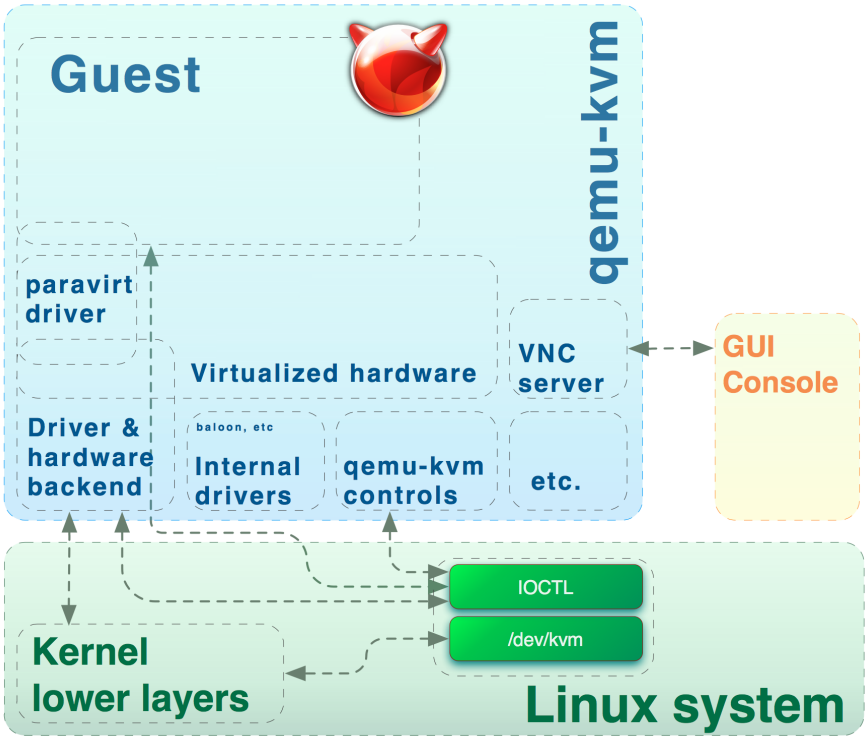


FIGURE 4.4: Conteneur qemu-kvm

La machine virtuelle Le système est contenu dans son propre conteneur à l'intérieur d'un processus qemu-kvm. L'espace mémoire dans lequel ce trouve ce conteneur est exécuté et géré par le périphérique /dev/kvm mais, il est directement lié au conteneur qemu-kvm. Les fichiers sources kvm-qemu/qemu-kvm-0.14.0/qemu-kvm.c, kvm-qemu/qemu-kvm-0.14.0/kvm/libkvm/libkvm.c et kvm-qemu/qemu-kvm-0.14.0/kvm-all.c permettent de comprendre comment ces interactions sont effectuées et comment ce conteneur et créé.

Le point important à retenir à propos de ces conteneurs est que bien que liés au contrôle du processus qemu-kvm, ce dernier doit passer par des ioctl et donc au travers de /dev/kvm pour interagir avec le fonctionnement de base de la machine virtuelle qu'il contient.

Matériel vritualisé, paravirutalisé et backend des drivers Comme le montre le graphique 4.4, le matériel virtualisé ainsi que le lien entre les périphériques paravirutalisés et les couches basses du noyau Linux sont réalisées en partie dans le processus /dev/kvm. Globalement, les fichiers en liens avec le hardware se trouvent dans le répertoire kvm-qemu/qemu-kvm-0.14.0/hw. Les fichiers liés à la paravirtualisation sont les fichiers nommés vitro-*.c.

Plus tard dans ce document, ces éléments sont expliqués plus en détail, mais le point à retenir est que dans ce cas précis, certaines opérations au travers de canaux sécurisés peuvent interagir plus rapidement avec le matériel (Couches basses du noyau Linux avec le passtrough) ou avec le matériel virtualisé (paravirtualisation).

Il n'est bien évidemment pas possible de décrire le fonctionnement détaillé de tous les composants hardware virtuels. Seulement, dans le cas de doute à propos du fonctionnement de l'un ou de l'autre, il est intéressant de les étudier, leur code étant en général bien structuré.

Contrôles et autres modules Cette partie du graphique représente les éléments qui permettent au processus de vivre en temps que tel et d'être contrôlé. Un point du code source où commencer à étudier ceci est la fonction kvm main loop cpu du fichier kvm-qemu/qemu-kvm-0.14.0/qemu-kvm.c. Cette fonction est la boucle de base d'un processus qemu-kvm. Il n'est cependant pas possible de lister tous les contrôles possibles dans ce

document.

Serveur VNC Bien qu'il existe un composant console kvm-qemu/qemu-kvm-0.14.0/console.c qui permette d'avoir une vue graphique d'une machine virtuelle, la plupart du temps, une connexion VNC est utilisée. Actuellement, qemu-kvm embarque un serveur VNC kvm-qemu/qemu-kvm-0.14.0/ui/vnc.c. Cette solution est la principale voie utilisée pour obtenir une interface graphique à une machine virtuelle.

Dans l'avenir, l'interface graphique ou la projection d'interface graphique avec une machine virtuelle se fera avec SIPICE [10] [11]. Il est donc intéressant de se documenter à propos de ce protocole qui deviendra bientôt un indispensable de la virtualisation.

Les chapitres précédents ont montrés de manière générale l'architecture de KVM et le fonctionnement de celui-ci. Le prochain sous-chapitre va tenter d'expliquer comment fonctionne le modèle d'exécution de KVM.

4.5 Architecture avancée, mémoire et contextes d'exécution

La modélisation présentée précédemment montre les parties du modèle KVM comme faisant ou non partie de l'espace noyau ou utilisateur. Ce présent chapitre essaie de montrer comment le noyau en fonction de ce qui doit être réalisé passe d'un mode à l'autre (Guest, Utilisateur ou Kernel). Ces modes ne doivent pas être confondu avec les modes CPU (Guest, Système) ou encore les niveaux de privilège CPU (Ring).

Avant d'entrer dans le modèle d'exécution d'une machine virtualisée sous KVM il est nécessaire de présenter les différents éléments exécutables dans un conteneur qemu-kvm. L'illustration 4.5 présente globalement ces différents éléments.

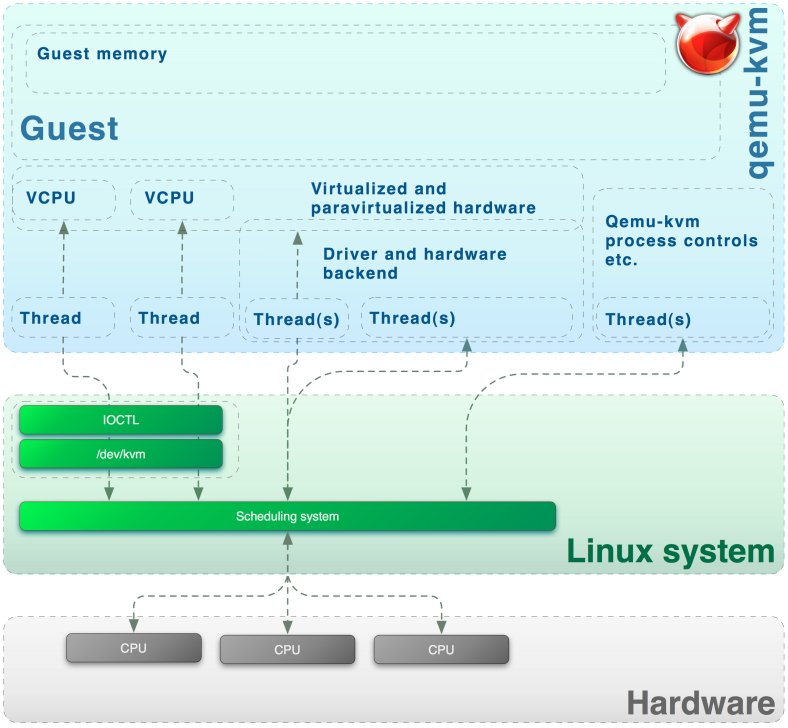


FIGURE 4.5: qemu-kvm-cpu

10. Page principale de SPICE : http://spice-space.org/
11. Présentation de RedHat à propos de SPICE : SPICE: An Open Remote Computing Solution : http://spice-space.org/docs/spice_redhat_summit_2009.pdf

- `Les processeurs virtuels` (VCPU) : Le mapping entre les processeurs virtuels et les processeurs réels est réalisé à l'aide de threads et des configurations dans /dev/kvm. Un thread par CPU virtuel fournit à la machine virtuelle. Le choix du processeur physique sur lequel s'exécute le processeur virtuel est laissé au scheduler de Linux. /dev/kvm s'occupe de gérer l'utilisation de l'assistance matérielle à la virtualisation. De ce fait, les threads qui réalisent le mapping des processeurs virtuels utilisent les ioctl's de /dev/kvm.
- `Périphériques paravirtualisés, périphériques virutalisés et backend des drivers` : Les périphériques paravirtualisés et virutalisés peuvent créer un certain nombre de threads. Par exemple : en cas d'utilisation de la pravirutalisation pour l'accès à des disques, la partie des périphériques virutalisée créera un certain nombre de threads au niveau du processus qemu-kvm ainsi qu'au niveau de la coucheAIO [12] de Linux. Par exemple, le driver pour les périphériques de type bloc se trouve dans le fichier kvm-qemu/qemu-kvm-0.14.0/hw/virtio-blk.c. Une fonction intéressante pour commencer son étude pourrait être virtio blk handle read.
- `Tâches internes au processus qemu-kvm` : Le processus qemu-kvm crée lui aussi un certain nombre de threads suivant les tâches qu'il a à réaliser.

Le modèle d'exécution d'une machine virtuelle concerne donc une partie des processus de virutalisation et aussi dans une certaine mesure, ceux des périphériques virtualisés et paravirutalisés. Dans ce chapitre, nous nous concentrerons sur la partie concernant les processeurs virtualisés. Un prochain chapitre expliquant les drivers vitrio et la paravirutalisation avec qemu-kvm s'attardera sur cette partie.

Tel qu'expliqué dans l'introduction de ce chapitre, le module KVM apporte un nouveau mode au noyau de Linux, le mode "guest". Ceci porte à trois (Guest, User, kernel) les modes que peut prendre le noyau de Linux. Suivant l'opération à réaliser, le noyau devra changer de mode pour la réaliser (par exemple : passer du mode Guest à User pour réaliser une écriture disque). Chacun de ces changements à un coup en temps et en ressources. Dans la documentation, les termes "Exit" ou "VM EXIT" sont utilisés pour présenter ce procédé (sortir du mode Guest).

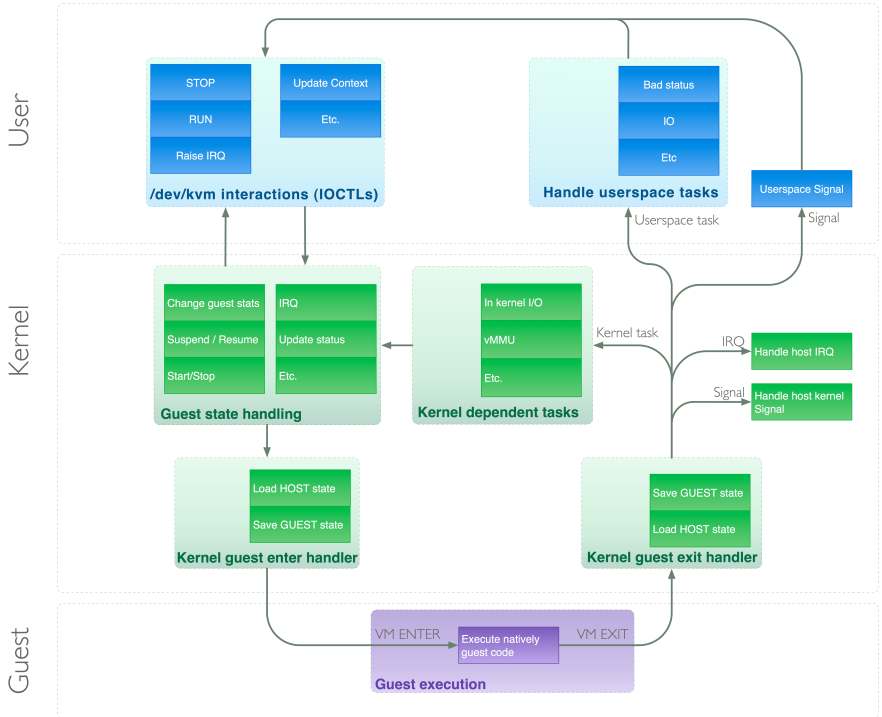


FIGURE 4.6: État d'exécution de KVM

La figure 4.6 présente de manière globale les différents flux d'exécution nécessaires pour le fonctionnement de

12. Article Wikipédia à propos des AIO : http://en.wikipedia.org/wiki/Asynchronous_I/O

KVM. Les termes présents en bas de l'image VM EXIT et VM ENTRY correspondent aux fonctions hardware permettant de forcer le passage et la sortie d'un processeur dans un mode de fonctionnement limité pour les machines virtuelles. Ces deux fonctions peuvent aussi représenter le passage en mode "Guest" du noyau. Le fichier linux/arch/x86/kvm/vmx.c contient les interactions avec l'assistance hardware à la virutalisation et donc la plupart de ces fonctions peuvent être étudiées directement depuis ce fichier. Cependant, il a été choisi de ne pas descendre à ce niveau pour le contenu de ce rapport.

Les deux catégories de sorties sont les "Lightweight VM EXIT" et les "Heavy-weight VM EXIT" . Les Lightweight VM EXIT sont les sorties du mode guest qui ne remontent pas jusqu'au mode "User" tendit que les Heavy-weight VM EXIT sont celles qui demandent de retourner en contexte utilisateur. Voici un exemple de déroulement avec quelques cas possibles de "VM EXIT" (Les Heavy et Light exits peuvent se combiner dans certains cas).

- 1 : Changement d'état du hardware (VM EXIT).
- 2 : Changement d'état du à dû software (IOCTL, IRQ, Kernel, etc).
- 3 : Analyse de la raison du changement de d'état.
- - : APIC Kernel : Lightweight VM EXIT
- - : IO APIC ou PIC Kernel Lightweight VM EXIT
- - : Manipulation mémoire du guest. Lightweight VM EXIT
- - : Réception d'un paquet réseau. Heavy-weight VM EXIT
- - : Écriture d'un bloc sur un disque. Heavy-weight VM EXIT
- - : etc.
- 4 : Gestion software du changement d'état.
- 5 : Changement d'état au niveau matériel (VM ENTRY).

Un changement d'état consomme en général plus de 10 000 cycles CPU (Approximation établie sur la base du code observé). Ces changements sont donc une des principales sources de perte de performances et ce, quelle que soit la solution de virtualisation choisie. Il est important de les minimiser et d'optimiser leurs réalisations. C'est d'ailleurs dans ce sens que la plupart des solutions de paravirtualisations tendent.

La section suivante de ce chapitre explique de manière générale comment la mémoire d'un processus qemu-kvm et de la machine virtuelle est allouée et de quoi il faut tenir compte lorsque l'on essaie de se représenter cette allocation mémoire.

4.5.1 Allocation de la mémoire par KVM et qemu-kvm

Un processus (conteneur) de machine virtuelle alloue et ce comporte comme un processus normal du point de vue de sa mémoire. Comme pour tout processus, la mémoire allouée en espace utilisateur (figure 4.7) est réservée de manière non linéaire dans la mémoire du système. Comme la figure suivante (4.7) le montre, la mémoire utilisée par le processus qui contient une machine virtuelle est allouée là où de l'espace est disponible dans la mémoire du système hôte.

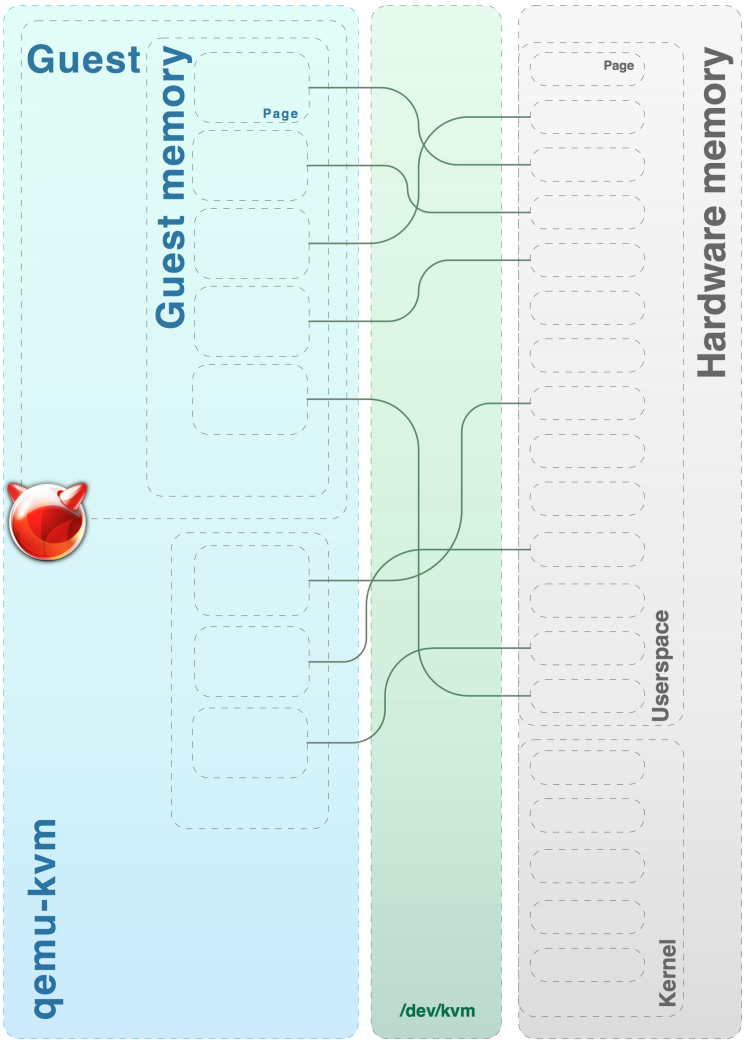


FIGURE 4.7: Allocation mémoire de KVM

Cette illustration (4.7) présente aussi un fait logique mais important, une machine virtuelle ne prend pas seulement l'espace mémoire qui est alloué à l'os invité. Il faut en effet tenir compte du fait que qemu-kvm consomme aussi de la mémoire pour son propre fonctionnement (Thread, process, buffers, matériel virtualisé, etc).

L'allocation mémoire et le cloisonnement sont réalisés au travers du périphérique /dev/kvm qui selon les cas et le support matériel disponible utilisera ou non les assistances matérielles.

4.6 Conclusion

Avec ce chapitre, il a été possible de passer en revue les principaux aspects de KVM. Cette solution de virtualisation apporte un réel vent de fraicheur au monde de la virtualisation par son approche d'implémentation profondément moderne et orientée vers l'ingénierie actuelle (réutilisabilité de ce qui est éprouvé mais, forte innovation). Le contrecoup d'une telle approche est un nombre de connaissances nécessaires accrues par rapport à des solutions concurrentes. Il est en effet nécessaire de bien comprendre certains mécanismes de Linux (par exemple le scheduler) pour être capable de maîtriser de manière pointue cette solution.

Les prochains chapitres consacrés à KVM vont passer en revues des éléments liés à cette solution ou considérés comme indispensables à l'utilisation de cette solution.

Sommaire

Chapitre

5

La paravirtualisation avec KVM

There is, though I do not know how there is or why there is, a sense of infinite peace and protection in the glittering hosts of heaven.

H. G. Wells, The Island of Doctor Moreau.

Le chapitre précédent a permis de voir l'architecture globale de KVM, le présent chapitre va maintenant expliquer comment est utilisé la paravirtualisation avec Virtio [1] dans qemu-kvm. La question de la paravirtualisation est un point essentiel pour des environnements de virtualisation efficaces en milieux réels. Une solution virtualisation ne peut en effet pas être efficace sans les performances apportées par cette technique.

De manière générale, virtio est une couche d'abstraction aux périphériques paravirutalisés. virtio a initialement été créé par Rusty Russell [2] comme solution de paravirtualisation pour son hyperviseur lguest [3] très proche voir parent de KVM. Aujourd'hui virtio est le drivier de paravirutalisation par défaut des Linux virtualisés. Il existe aujourd'hui aussi un portage Windows de virtio.

La première partie de ce chapitre présentera une approche simplifiée de virtio puis, tout comme pour KVM et qemu-kvm l'architecture de virtio sera disséquée et commentée pour en permettre la compréhension générale.

1. Page officiel des drivers virtio : http://wiki.libvirt.org/page/Virtio
2. Article Wikipédia à propos de Rusty Russel : http://en.wikipedia.org/wiki/Rusty_Russell
3. Article Wikiédia à propos de lguest : http://en.wikipedia.org/wiki/Lguest

5.1 Vision abstraite de virtio

Tel que présenté en introduction, virtio est une couche d'abstraction qui a pour but de donner au système invité l'accès à un panel défini de périphériques paravirtualisés disponibles sur l'hyperviseur au travers d'une interface commune. La figure ci-dessous 5.1 présente pourquoi une telle couche d'abstraction est un élément essentiel dans le monde de la virtualisation. Ces drivers rendent en effet la partie installée sur le système invité compatible avec toute solution de virtualisation qui implémentera un "backend" compatible virtio. De plus, l'utilisation d'une base commune permet un développement en commun aux bénéfices de tous les participants (pour les versions open-source des implémentations de virtio). Il est en effet possible de faire fonctionner les drivers virtio sur des backends qui ne sont pas des backends spécialement étudiés pour virtio mais qui en respectent les spécifications d'interfaçâges avec le système invité.

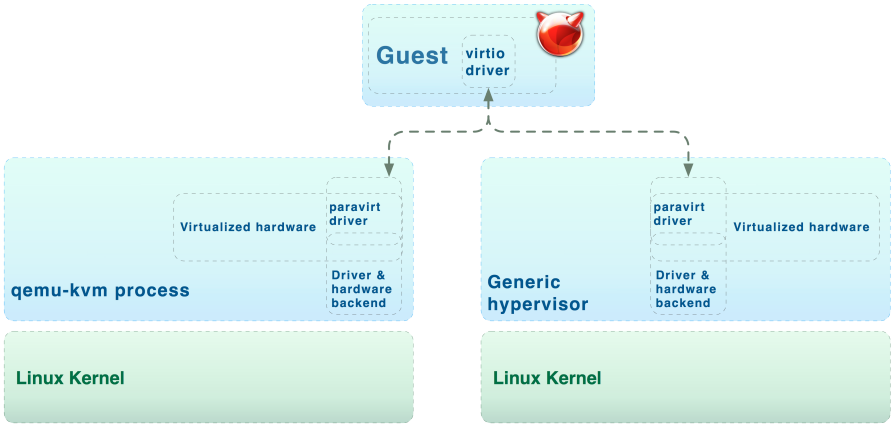


FIGURE 5.1: Abstraction des drivers avec virtio.

Un point intéressant avec KVM et d'autres implémentations de virtio est que les drivers se trouvant dans le système invité communiquent avec un backend qui se trouve en espace utilisateur. Donc, les actions d'écritures et de lectures sont réalisées à partir de cet espace (utilisateur) ce qui rend de grands services en termes de sécurité et d'isolation. De manière générale, comme ce sera discuté dans les sections suivantes, la communication se passe par abstraction d'un tampon entre les deux environnements (invité et hôte) qui tout en étant simple est un principe très efficient.

5.2 Architecture de virtio

Globalement, virtio comporte donc deux parties principales. Le driver "front-end" installé dans le système invité et le "backend" installé dans l'hyperviseur. De plus, virtio propose deux couches chargées de la communication entre le système invité et l'hyperviseur. La figure 5.2 présente globalement l'architecture de virtio. En haut du graphique se trouve une file appelée Transport. Cette file sert conceptuellement de connecteur entre le "front-end" et le "back-end" des drivers virtio. Techniquement, en fonction des besoins, les drivers pourront créer et utiliser plusieurs queues. Par exemple, dans le cas des drivers réseaux (virtio network) deux queues sont utilisées pour ce driver (l'une pour la réception et une autre pour l'émission) tendis que les drivers pour les périphériques de type bloc n'en utilise qu'une seule.

D'un point de vue technique, la plupart des implémentations de virtio utilise un tampon circulaire pour créer le pont entre invité et hôte.

Comme le montre l'image 5.2, virtio propose un certain nombre de périphériques. La liste ci-dessous propose de passer rapidement en revue les principaux et de donner un pointeur sur le code source du noyau Linux où ils se trouvent.

Les éléments de communication entre le système invité et le système hôte :

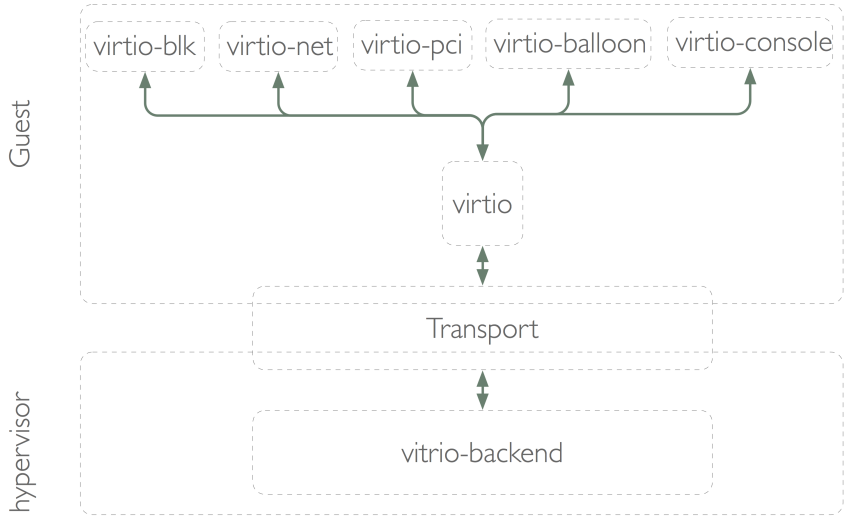


FIGURE 5.2: Architecture logiciel des drivers virtio

- `virtio` : ./linux/dirvers/virtio/virtio.c : Composant de communication qui contient les fonctionnalités de présentation des périphériques virtuels au système et leurs liens avec le tampon de communication avec le système hôte.
- `Transport` : ./linux/drivers/virtio/virtio ring.c : Tampon de communication avec le système hôte.

les principaux périphériques :

- `virtio blk` : ./driver/block/virtio blk.c : Périphérique/driver chargé du traitement des périphériques de type bloc comme par exemple : les disques durs.
- `virtio net` : ./driver/net/virtio net.c : Périphérique/driver chargé du traitement des périphériques de type cartes réseaux.
- `virtio pci` : ./driver/vitrio/virtio pcl.c : Ce driver est chargé des aspects lié aux bus PCI et à la connexion de device en "passthrough".
- `virtio balloon` : ./driver/virtio/balloon.c : Ce driver se charge de permettre l'application du ballooning (gestion dynamique de la mémoire du système invité) dans les systèmes invités.
- `virtio console` : ./driver/virtio/virtio console.c : Driver pour l'utilisation d'une console.

D'un point de vue conceptuelle, chaque driver en "front-end" dispose d'une implémentation correspondante en "back-end".

Donc, pour le cas de qemu kvm, Le backend du tampon de communication ce trouve dans :

- `virtio` : qemu kvm/hw/virtio.c : Récupère les communications depuis les tampons et les redistribue aux bons périphériques du backend.

Pour les périphériques :

- `virtio blk` : qemu kvm/hw/virtio balloon.h
- `virtio net`: qemu kvm/hw/virtio net.c
- `virtio pci` : qemu kvm/hw/virtio pci.c
- `virtio balloon` : qemu kv/hw/virtio balloon.c
- `virtio console` : qemu kvm/hw/virtio console.c

La section suivante s'intéresse un peu plus à l'architecture logicielle proposée par ces drivers.

5.2.1 Communication par hiérarchie

Pour mieux comprendre les principes de communications, plaçons-nous du côté du système invité. De ce point de vue (tel que montré sur la figure 5.3), le sommet du driver virtio représente le front-end des drivers pour le système invité.

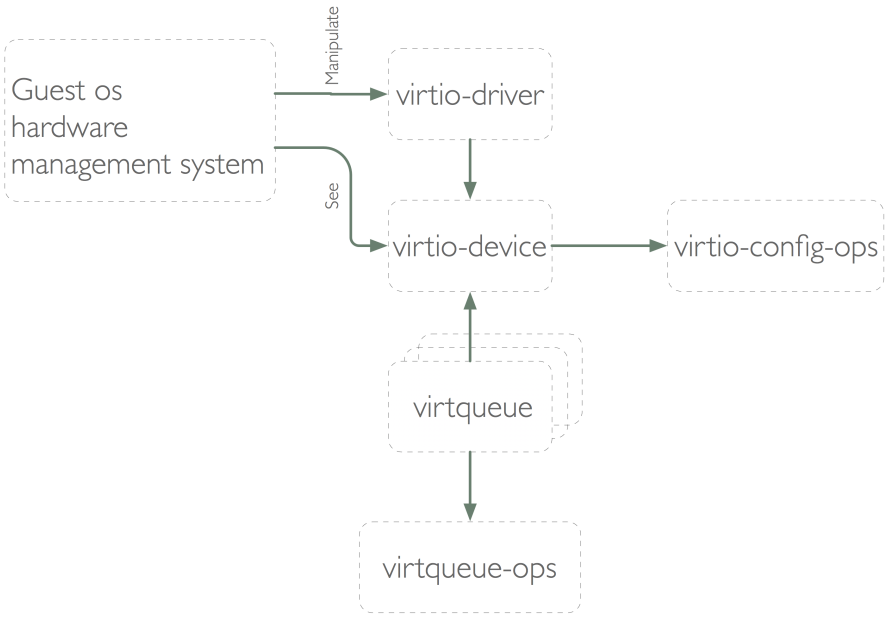


FIGURE 5.3: Stack logiciel de la communication avec virtio

Les `virtio drivers` (Définit dans ./linux/include/linux/virtio.h ligne 127) sont donc une couche qui permet au système invité d'utiliser les périphériques (`virtio devices`). les `virtio devices`(Définit dans ./linux/include/linux/virtio.h ligne 101) sont eux-mêmes une représentation de périphériques qui se trouvent dans le "back-end" du système hôte. Ils présentent donc les périphériques au système invités. Cette présentation est liée aux `virtio config ops` qui définissent les opérations nécessaires pour configurer les périphériques virtio.

Chaque périphérique (`virtio devices`) est lié à une ou plusieurs files (`virtqueue`). Chacune de ces files (`virtqueue` (Définit dans ./linux/include/linux/virtio.h ligne 20)) contient une référence vers le périphérique qui l'utilise. Finalement, chaque file est liée à l'objet `virtqueue ops` (Les différentes opérations sont définies dans ./linux/include/linux/virtio.h lignes 28 à 90) qui définit les opérations de bas niveau pour la communication avec le "back-end" des drivers qui se trouvent sur le système hôte.

Maintenant que les différents éléments ont été décrit, il est possible d'expliquer ce qu'il se passe pratiquement.

5.2.2 Ajout d'un périphérique

Tout commence par la création d'un `virtio driver` et son enregistrement (auprès du backend) avec la fonction register virtio driver (./linux/drivers/virtio/virtio.c ligne 173) qui va permettre au périphérique et aux drivers d'utiliser le bus virtio. La définition des périphériques `virtio driver` contient entre autre : Le driver à utiliser, la liste des périphériques qu'il doit permettre d'utiliser, une liste des fonctionnalités (varie selon le périphérique) et une liste de fonctions permettant d'avoir accès à ce driver.

Quand l'hyperviseur identifie la présence d'un périphérique qu'il reconnait, l'hyperviseur appel le driver et lui transmet le périphérique qu'il devra utiliser. En cas de besoin, des fonctions de types `virtio config ops` peuvent être appelées pour terminer la configuration du périphérique.

Le périphérique ne contient pas de référence à des éléments de type `virtqueue` (mais, les `virtqueue` ont des références sur les périphériques). Pour identifier une `virtuqueue` associée à un périphérique, il est nécessaire d'utiliser les fonctionnalités de `virtio config ops`. Ces fonctions permettront de retrouver la file associée à l'instance du périphérique utilisé. La file (`virtqueue`) possède quant à elle un lien qui lui permet de notifier le système invité quand une réponse lui est destinée dans le tampon.

Une `virtqueue` est donc simplement un élément qui possède : une fonction dite de "callback" qui est utilisée quand l'hyperviseur lit le tampon, une référence au périphérique lié, une référence aux opérations possibles sur cette file et une référence vers l'implémentation back-end avec laquelle communiquer.

Le cœur de cette architecture est fondamentalement les `virtqueue` qui définissent quand et comment les données doivent être déplacées entre le système invité et le système hyperviseur.

5.2.3 Utilisation du tampon

Les drivers qui se trouvent dans le système invité vont donc communiquer avec le backend des drivers se trouvant du coté hyperviseur au travers d'un ou des buffers. Suivant les fonctionnalités, le système invité va proposer un ou plusieurs buffers à lire et écrire pour traiter les différentes requêtes.

Par exemple, imaginons un cas avec trois tampons. Dans ce cas nous pourrions utilisé 1 tampon pour créer des requêtes de lecture (rempli par le guest mais, lu par l'hyperviseur) et deux tampons (rempli par l'hyperviseur mais, lu par le guest) pour recevoir les réponses.

5.2.4 Présentation de la virtqueue cœur de ce système

Les sections précédentes ont montré qu'un lien entre le système invité et le système hôte pouvait se faire grâce aux périphériques virtio paravirtualisés, mais plus précisément grâce aux queues `virtqueue` qu'ils utilisent.

`virtqueue` est construit comme une API à part (bien que relativement réduite). Cette API est composée de 5 fonctions principales.

La première fonction add buf (Fichier ./linux/drivers/virtio/virtio ring.c:virtqueue add buf gfp ligne 159) est utilisée pour envoyer une requête vers l'hyperviseur. Cette requête prend en général la forme d'une liste d'adresses et de longueurs (adresses où trouver les données de la requêtes et longueur de ces données). Pour utiliser cette fonction, le système invité doit fournir à la fonction un certain nombre de paramètres. Ces paramètres sont : la ou les `virtqueue` concernées, la requête, le nombre de tampons utilisés dans la requête (tableau précédemment cité) et le nombre de tampons pour écrire la réponse.

Une fois que le système invité à créer la requête (liste de buffers), il doit notifier l'hypervieur à l'aide de la fonction kick (Fichier ./linux/drivers/virtio/virtio ring.c:virtqueue kick ligne 237). Plus le système invité attend avant de notifier l'hyperviseur, moins il y aura de perte de performance provoqué par les changements d'état, mais comme il n'est pas possible de ne faire qu'attendre, le dimensionnement et le nombre de ces tampons sont les clés des performances des drivers virtio et de la paravirtualisation en général.

La réponse de l'hypervieur à une requête se récupère au travers de la fonction get buf (Fichier ./linux/drivers/virtio/virtio ring.c:irtqueue get buf ligne 289). Le système invité a deux moyens pour savoir qu'une réponse est arrivée. Il peut continuellement appeler cette fonction (polling) ou alors en utilisant la fonctionnalité de "callback" présente dans la `virtqueue` l'hyperviseur peut notifier le système invité d'une réponse.

Les deux dernières fonctions sont `enable cb` (Fichier /linux/drivers/virtio/virtio ring.c:virtqueue enable cb ligne 340) et `disable cb` (Fichier ./linux/drivers/virtio/virtio ring.c:virtqueue disable cb ligne 332). Ces deux fonction servent à activer ou désactiver la possibilité de "callback" (Notification du système invité par l'hyperviseur).

De manière générale, le contenu des tampons utilisés et le choix de leurs formats sont la responsabilité du périphérique virtualisé et de son "backend" côté hyperviseur. Les couches de transport comme les `virtqueues` n'ont aucune connaissance du contenu qu'elles transportent.

Pour les personnes plus intéressées par l'implémentation de virtio, un document [4] écris par IBM en 2008 est disponible et très riche en détails techniques avancés. Bien que ce document ne soit plus d'actualité, les principes expliqués le sont encore.

5.3 Étude de cas, le driver network

Maintenant que l'architecture global des drivers virtio a pu être passée en revue, il peut être intéressant de voir plus en détails à un de ces périphérique en comme exemple. Ce chapitre concerne le périphérique réseau et essaiera de rester superficiel du point de vue de son code. L'objectif est de montrer les principes mis en oeuvre pour gagner en performance et pas forcement chaque partie du code de ce périphérique.

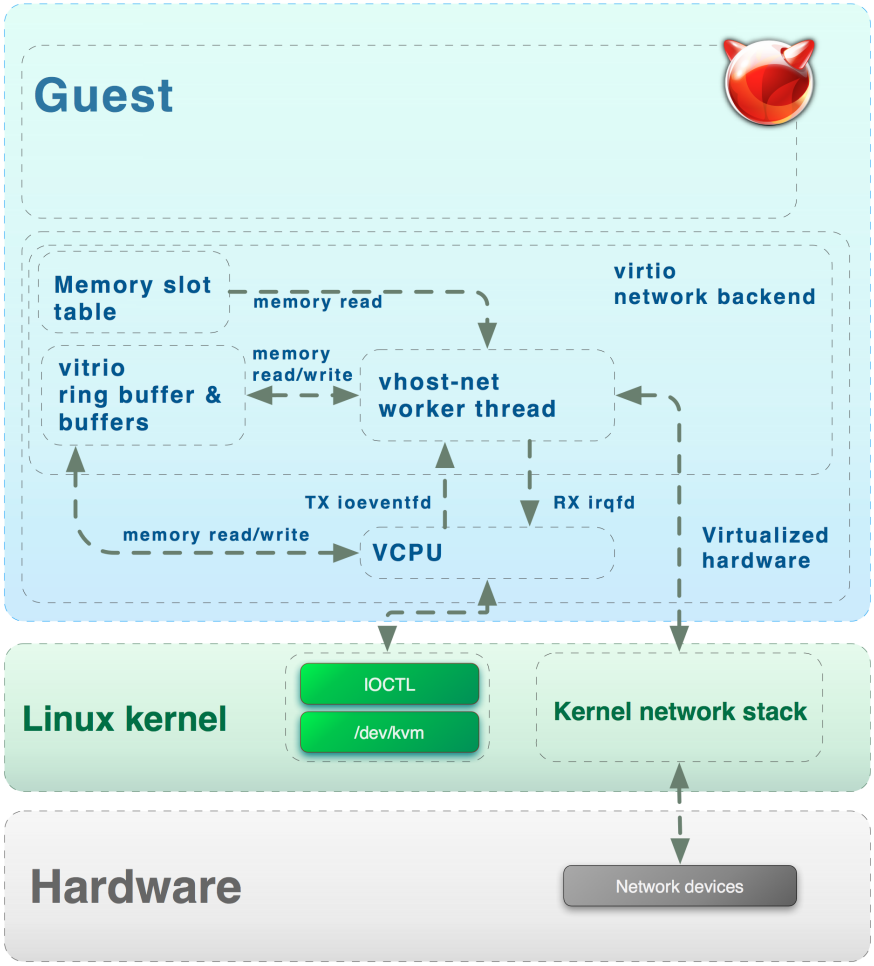


FIGURE 5.4: Plan du driver réseau virt-net

La figure 5.4 présente l'architecture de la carte réseau virtio. Globalement, ce design cherche à répondre à trois objectifs :

- Éliminer autant que possible les "heavy exit" qui font perdre beaucoup de temps cpu.
- Réduire au maximum la copie de paquets (D'un tampon à un autres, d'une file à une autre, d'un processus à un autres, etc) pour éviter les pertes de temps et le gaspillage mémoire.

4. virio: Towards a De-Facto Standard For Virutal I/O Devices : http://ozlabs.org/~rusty/virtio-spec/virtio-paper.pdf

- Empêcher toutes actions en mode kernel et maintenir les actions en espace utilisateurs (process qemu) au tant que possible.

Pour réussir ceci, l'architecture de ces drivers de base utilise différents composants.

En espace utilisateur, des kernel helpers[5] (Outil mise à disposition par le noyau Linux pour rendre le développement plus facile par exemple : des listes et des piles manipulées avec l'aide du noyau) sont définis et configurés.

Comme pour tout driver virtio, l'interface host-guest est réalisée à l'aide des mécanismes proposées par les virtqueue (virtio ring.buffer sur le graphique).

Un thread de fonctionnement (workerthread) s'occupe de traiter le remplissage et le vidage des piles pour permettre la meilleure optimisation possible.

L'interface eventfd[6] (permet de créer des notifications entre processus utilisateur et entre le noyau et les processus utilisateur dans le cadre de mécanismes d'attente) de KVM. Cette interface est utilisée pour gérer les deux types d'événement.

- `TX trigger via` eventfd : Événement généré par le système invité pour indiquer à l'hyperviseur que des paquets doivent être envoyés.
- `RX signal via` irqfd : Événement généré par l'hyperviseur (backend du driver) pour prévenir le système invité que des paquets sont disponibles à la lecture.

Le dernier élément est la possibilité d'attacher le périphérique virtuel à une interface réseaux via les TAP et les macvtap ou des bridges Linux.

Avec l'aide de cette architecture minimisant les changements de mode du noyau, les drivers de paravirutalisation réseaux peuvent, suivant les backends, proposer des "throughput" de plusieurs gigabytes.

5. Linked Lists and Work Queues : http://www.linux-mag.com/id/2518/
6. How-to evenfd : http://www.linuxhowtos.org/manpages/2/eventfd.htm

5.4 Conclusion

Ce chapitre a présenter de manière globale l'architecture des drivers paravirtualisés virtio utilisés dans de nombreux hyperviseurs et standard en devenir. Ce chapitre est revenu sur les principales composantes de ces drivers ainsi que sur les mécaniques misent en oeuvre.

Cette architecture est construite sur de bonnes bases et les possibilités d'évolutions sont encore nombreuses. Par exemple :

- Utiliser les mécanismes de "Transparent huge pages" [7].
- Optimiser les mécanismes reposant sur des architectures NUMA(Non-Uniform Memory Access) [8]. par exemple, pour le Scheduling.
- Améliorer la gestion des "Spin Locks" [9] [10] (mécanisme de gestion de la concurrence utilisé par exemple pour la gestion des drivers) et leurs effets lors de préemptions.
- Continuer d'éliminer le maximum de copies de paquets possibles.
- Améliorer la gestion de la répartition des tâches liéés aux devices dans le noyau de Linux.
- Proposer du matériel capable de gérer la virtualisation comme les nouveaux CPU.
- etc.

Le prochain chapitre s'intéresse à décrire un élément fondamental d'un système de virtualisation basé sur kvm, la librairie libvirt.

7. Transparent huge pages in 2.6.38 : https://lwn.net/Articles/423584/
8. Article Wikipédia à propo du NUMA : http://en.wikipedia.org/wiki/Non-Uniform_Memory_Access
9. Using Spin Locks: An Example : https://www.osronline.com/ddkx/kmarch/synchro_8f1j.htm
10. Publication de AMD sur la gestion des "Spin Locks" : https://www.amd64.org/fileadmin/user_upload/pub/2008-Friebel-LHP-GI_OS.pdf

Sommaire

Chapitre

6

libvirt

The hard must become habit. The habit must become easy. The easy must become beautiful.

Doug Henning quotes (Canadian Magician).

La librairie Libvirt [1] est une API Linux d'interface avec les capacités de virtualisation de Linux. Cette interface est actuellement compatible avec de nombreux hyperviseurs [2](par exemple, Xen, Qemu ou encore KVM).

Lorsque des systèmes de virtualisation atteignent une certaine taille (Par exemple : des clouds, des datacentersvirtualisés, etc.) il devient important de pouvoir les administrer de manière simple et uniformisée. D'autant plus si ces différentes infrastructures reposent sur des technologies de virtualisation différentes. Libvirt va permettre d'administrer et de manipuler les systèmes hébergé spar les systèmes de virtualisation. De part sa nature d'API, libvirt ne propose pas directement des outils, mais une panoplie d'interfaces et de fonctions pour interagir avec les différents hyperviseurs.

Aujourd'hui, libvirt est un composant indispensable pour utiliser un environnement KVM. Ce chapitre va donc décrire cette librairie et comment l'utiliser.

1. Site officiel de libivrt : http://libvirt.org/
2. Liste des drivers pour hypveriseur : http://libvirt.org/hvsupport.html#virDriver

6.1 Architecture

Commençons la découverte de libvirt par son architecture et plus précisément par son modèle (6.1).

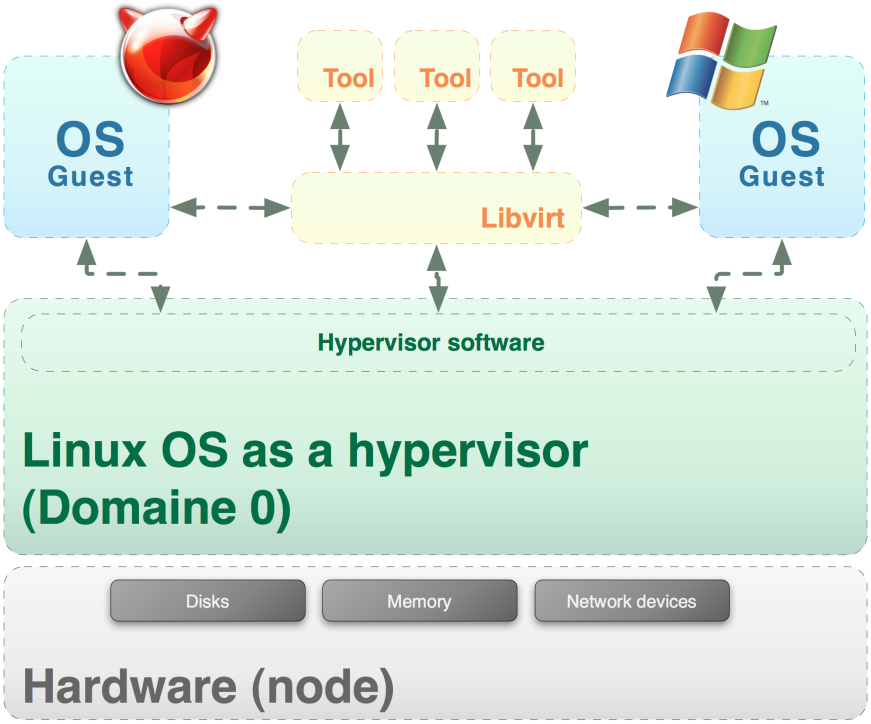


FIGURE 6.1: Archi général d'un système équipé de libvirt

Comme le montre la figure 6.1, libvirt permet de maîtriser plusieurs hyperviseurs mais, apporte aussi un vocabulaire particulier. Cette terminologie est relativement importante parce qu'elle est utilisée dans toute la documentation officielle de libvirt. Sur cette image, un "Node" représente l'élément physique hébergeant un ou plusieurs hyperviseurs, le "Domaine 0" est le système faisant fonctionner un ou plusieurs hyperviseurs tendis que les "Domaines" sont eux les systèmes virutalisés.

Un point important et que libvirt est cantonnée à l'espace utilisateur.

6.1.1 Support des hyperviseur

Comme le montre la figure 6.2, l'interface entre les hyperviseurs et libvirt se fait au travers de drivers. Ces drivers permettent de faire le lien entre les fonctions génériques utilisées par libvirt et celles des hypervieurs gérés [3].

Il arrive donc que certaines fonctions ne soient pas supportées par les hyperviseurs tout comme certaines particularités des hyperviseurs ne sont pas supportées par libvirt.

Sur la figure (6.2), il est possible de voir le démon libvirtd. Ce démon permet d'obtenir un accès distant et local au domaine0 et aux fonctionnalités de libvirt dont il dispose.

3. Driver libvirt actuel : http://libvirt.org/hvsupport.html

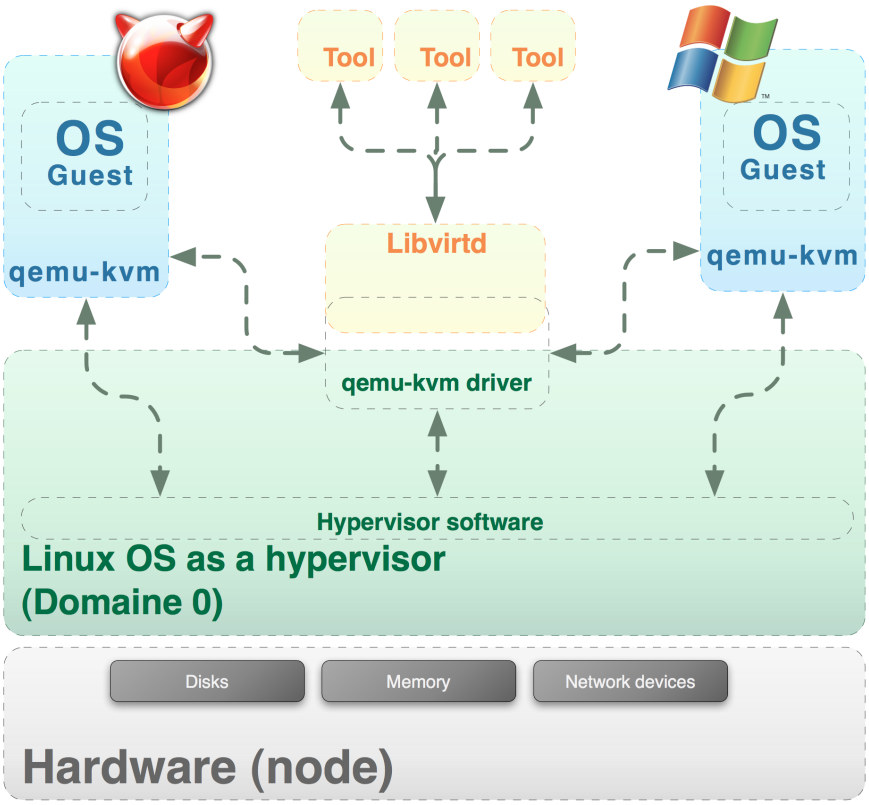


FIGURE 6.2: Interaction entre libvirt et un hyperviseur

6.1.2 Contrôle et gestion distantes

Libvirt permet deux modes de contrôle, le premier, est d'utiliser directement l'API libvirt en se trouvant sur le même node que les domaines à gérer. Dans ce cas, les applications utilisant libvirt vont directement utiliser les couches proposées par libvirtd. Le second moyen est d'utiliser des applications de gestion qui se trouvent sur un autre node. Pour ce second cas, une connexion à distance doit être réalisée (voir figure 6.3).

Pour réaliser cette communication, le démon libvirtd est utilisé. Ce démon est démarré automatiquement quand libvirt est installé sur un node et détecte de manière automatique quels sont les drivers à utiliser pour interagir avec les divers hyperviseurs présents. Cette communication se fait au travers un canal réseau (en général SSH) avec le système distant.

6.2 Présentation générale de l'API

L'API libvirt peut-être découpée en 5 grandes parties : "hypervisor connexion API", "domain API", "network API", "storage volume API" et "storage pool API".

Toutes les communications utilisées par libvirt ont lieu après une connexion avec l'hyperviseur. L'API "hypervisor connexion API" permet de créer les connections pour toutes les autres API. En général, cette connexion se fera au travers d'une fonction de connections et retournera un élément de référence vers cette connexion. Par exemple, en C, la fonction " virConnectOpen" [4] créera la connectivité et retournera un élément du type "virConnectPtr" [5] qui est la référence sur un objet ou élément représentent la connexion. Cet élément ou objet est l'élément utilisé par toutes les autres fonctionnalités de management.

4. http://libvirt.org/html/libvirt-libvirt.html#virConnectOpen
5. http://libvirt.org/html/libvirt-libvirt.html#virConnectPtr

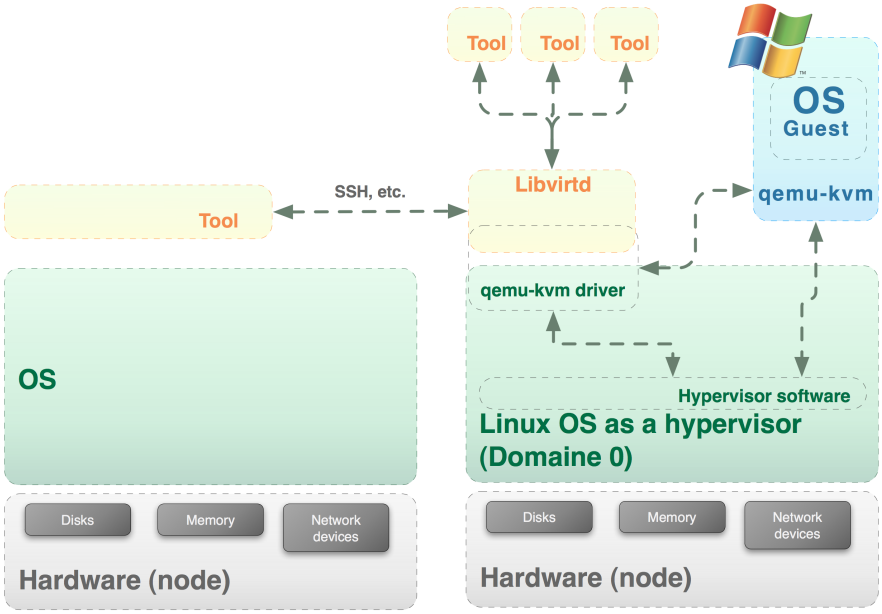


FIGURE 6.3: Confection à distance sur libvirt

En général, pour obtenir la liste des capacités de chaque hyperviseur, il est possible d'utiliser des fonctions pour les récupérer. En C, la fonction "virConnectGetCapabilities" [6] retournera les capacités de l'hyperviseur et la fonction "virNodeGetInfo" [7] permettra de récupérer des informations sur le node. Dans ces deux cas, les informations sont retournées au format XML.

Une fois la connexion avec l'hyperviseur établie, il est possible de manipuler les différentes ressources disponibles. Par exemple : avec la fonction "virConnectListDomains" [8], il est possible d'obtenir la liste des domaines disponibles. Cette fonction retournera la liste des domaines disponibles sur les différents hyperviseurs connectés.

Avec l'aide de l'API "domain API", il est possible de réaliser de très nombreuses opérations sur les domaines ou de les manipuler. D'autres fonctionnalités de ces API permettent aussi d'inspecter et gérer la notions de réseau "Network API" ou encore de ressources ("storage volume API" et "storage pool API"). Ces API supportent également des mécanismes permettant de créer des notifications en fonctions événements intervenants sur les domaines observés. Comme par exemple : Boot d'un domaine, suspension d'un domaine, etc.

6.2.1 Compatibilité de l'API

Les bases de libvirt ont été implémentée en C (avec support du c++) et incluent de manière native un support pour le langage python (langage de prédilection pour le développement de tools). Cependant, il existe aussi un support pour de très nombreux langages tels que : Ruby, Java, Perl, OCaml (langage utilisé pour les outils de la famille des virt-tools (virt-top par exemple)) , etc.

6.2.2 Exemple avec python

Cette section présente un petit exemple de connexion à un hyperviseur local avec python pour ressortir des informations à propos de l'état de différents domaines.

```
import libvirt

#establishment of a connection with the local hosted hypervisor (qemu)
```

6. http://libvirt.org/html/libvirt-libvirt.html#virConnectGetCapabilities
7. http://libvirt.org/html/libvirt-libvirt.html#virNodeGetInfo
8. http://libvirt.org/html/libvirt-libvirt.html#virConnectListDomains

```
hyp_conn = libvirt.open('qemu:///system')

#for all the domains fouds
for id in hyp_conn.listDomainsID():

    #get the curent domaine
domain = hyp_conn.lookupByID(id)

    #print some information
print "domain %s  State %s" % ( domain.name(), domain.info()[0] )
    #suspend then print some informations
domain.suspend()
print "domain %s  State %s (after suspend)" % ( domain.name(), domain.info()[0] )
    #resume then print some informations
domain.resume()
print "domain %s  State %s (after resume)" % ( domain.name(), domain.info()[0] )
```

Cette exemple commenté produira le résultat suivant :

```
domaine 1-Fedora-14-64bits  State 1
domaine 1-Fedora-14-64bits  State 3 (after suspend)
domaine 1-Fedora-14-64bits  State 1 (after resume)
domaine 2-Fedora-14-64bits  State 1
domaine 2-Fedora-14-64bits  State 3 (after suspend)
domaine 2-Fedora-14-64bits  State 1 (after resume)
domaine 4-Fedora-14-64bits  State 1
domaine 4-Fedora-14-64bits  State 3 (after suspend)
domaine 4-Fedora-14-64bits  State 1 (after resume)
domaine 3-Fedora-14-64bits  State 1
domaine 3-Fedora-14-64bits  State 3 (after suspend)
domaine 3-Fedora-14-64bits  State 1 (after resume)
domaine 5-Fedora-14-64bits  State 1
domaine 5-Fedora-14-64bits  State 3 (after suspend)
domaine 5-Fedora-14-64bits  State 1 (after resume)
```

Ce petit scripte permet de voir qu'il est très simple de communiquer avec un hypervisuer via libvirt ce qui rend la construction de scriptes et d'outils de maintenance aisée.

6.2.3 Les applications qui utilisent libvirt

Ce chapitre n'a montré qu'un petit spectre de ce que peut réaliser libvirt. Mais, cela est suffisant pour se rendre compte que les possibilités d'utilisation de cette librairie sont réellement étendues.

Il existe actuellement une grande quantité d'applications qui utilisent libvirt et qui peuvent actuellement être utilisées comme outils de maintenance de systèmes de virtualisations. Par exemple :

- `virsh`[9] : Console d'administration permettant d'utiliser la plupart des fonctionnalités de libvirt
- `virt-install` : Outil permettant la création de systèmes invités et leurs installations.
- `virt-viewer` : Solution sécurisée et légère pour attacher une console graphique à une machine virtuelle.
- `virt-clone` : Solution pour le clonage de systèmes invités.
- `Ovirt`[10] : Outils de gestion et de manipulation de cloud composé de centaines à plusieurs milliers de machines virtuelles.
- etc.

9. irsh Command Reference : http://libvirt.org/sources/virshcmdref/html-single/
10. Page principale du projet ovirt : https://fedorahosted.org/ovirt/

6.3 Conclusion

libvirt est clairement un outil extrêmement puissant qui permet de réaliser la quasi-totalité des tâches de maintenance liées aux environnements de virtualisation. Dans ce chapitre, nous avons vu l'architecture de libvirt ainsi qu'un petit extrait de son API.

Ce chapitre conclut l'analyse de la solution KVM. Pour des raisons de temps, il n'aura pas été possible d'aller plus loin dans ce rapport bien qu'en pratique, une grande quantité d'informations a pu être récoltée.

Les prochains chapitre présentent quelques éléments pratiques de l'architecture KVM.

III

KVM
Aperçu de mise en œuvre

Sommaire

Chapitre

7

Installation de l'environnement KVM

An ounce of practice is worth more than tons of preaching.
Mohandas Gandhi.

Maintenant que l'environnement KVM a été décrit dans les chapitres précédents, il est intéressant de passer à la pratique. Ce chapitre présentera quels sont les éléments à installer sur un système Linux pour être capable d'utiliser la solution de virtualisation KVM.

Bien que les démonstrations seront menées sur une distribution Linux [1] unique, les explications fournies sont applicables avec toutes les distributions supportant KVM ou les logiciels présentés. Le choix de se limiter à une seule distribution a été guidé par les affinités de l'auteur de ce document ainsi que par la disponibilité rapide de certains logiciels sous forme de paquetage.

Ce chapitre n'est pas un guide d'installation de KVM, mais il vise à montrer différentes possibilités d'installation de KVM dans un environnement Linux. La documentation [2] officielle de `RedHat` décrit de manière très claire l'installation manuelle d'un environnement KVM.

La seconde partie de ce chapitre s'intéressera aux moyens d'automatiser l'installation d'un système dédié à la virtualisation (qu'il soit muni ou non d'une interface graphique). Aujourd'hui, il existe de nombreuses méthodes pour installer un système d'exploitation de manière automatique. Celle qui sera utilisée dans le cadre de ce projet permet au choix d'obtenir, un système complètement à jour, ou, avec une partie ou tous les paquets dans des versions bien définies. De la même manière, la méthode utilisée permettra de n'avoir qu'un seul dépôt ou aucun (en se reposant sur les dépôts existants). Le troisième avantage de la solution utilisée est de permettre une maintenance et une personnalisation de ce système de déploiement de manière simple et efficace.

La dernière partie de ce chapitre posera les spécifications des futures installations qui devraient être réalisées dans les laboratoires de HEPIA. Le temps ne permettant pas de le mettre directement en pratique dans ce travail, ces futures installations seront spécifiées et de larges explications et références sur les technologies à employer seront données au travers de cette fin de chapitre.

1. Article Wikipédia à propos de la notion de distribution Linux : http://en.wikipedia.org/wiki/Linux_distribution
2. Documentation "Virtualization" de RedHat : https://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization/index.html

7.1 Sélection du système d'exploitation

Pour ce travail, le choix de la distribution utilisée s'est arrêté sur Fedora 14. Ce choix a été motivé par plusieurs raisons :

- Paquets à jour.
- Les paquets sont tous livrés avec leurs règles SELinux (voir boîtes d'informations) correspondantes.
- Connaissances générales de l'environnement par l'auteur.
- Qualité de la documentation officielle et non officielle disponible.
- Comptabilité avec l'une des familles de distribution la plus utilisée dans l'environnement professionnel (RedHat).

SELinux :

Security-Enhanced Linux, abrégé SELinux, est un Linux security module (LSM), qui permet de définir une politique de contrôle d'accès obligatoire aux éléments d'un système basé sur Linux.

Le projet SELinux a été initié [3] et spécifié [4] par la NSA [5] [6] pour répondre à ses besoins en terme de sécurité multiniveau.

Son architecture dissocie l'application de la politique d'accès et sa définition. Il (SELinux) permet notamment de classer les applications d'un système, en différents groupes, avec des niveaux d'accès plus fins. Il permet aussi d'attribuer un niveau de confidentialité pour l'accès à des objets systèmes, comme par exemple : des descripteurs de fichier, des utilisateurs ou encore des processus. L'accès aux objets est géré selon un modèle de sécurité multiniveau (MLS pour Multi level Security). SELinux utilise le modèle Bell LaPadula [7] avec "Type enforcement" de SCC [8] pour l'intégrité.

De manière générale, SELinux doit être perçu comme un modèle de sécurité de type Mandatory Access Control(MAC) [9] permettant de créer des cloisonnements à tous les niveaux du système Linux. [10] [11]

LSM (Linux Security Modules) :

Linux Security Modules (LSM) est une infrastructure qui permet au noyau Linux de prendre en charge divers modèles formels de sécurité, ce qui évite de favoriser une implémentation de sécurité particulière parmi toutes celles existantes. [12].

Fedora [13] est une distribution moderne dont chaque version (pour les paquets de base) ne possède qu'un support de 2 versions [14] + 1 mois. Une fois cette période écoulée, les patchs et les correctifs ne sont plus maintenus pour les versions précédentes. Par exemple :

- Fedora 13 n'est plus supportée 1 mois après la sortie de la 15
- Fedora 14 n'est plus supportée 1 mois après la sortie de la 16

Une version de Fedora sortant environ tous les 6 mois, cela donne un support d'environ 1 ans.

Il existe cependant un dépôt de paquets Fedora-EPEL [15] qui permet à Fedora de fournir un support long

3. Historique de la création de SELinux par la NSA : http://www.opensourceforamerica.org/case-studies/nsa
4. Spécifications de SELinux fournies par la NSA : http://www.nsa.gov/research/selinux/
5. Article Wikipédia à propos de la NSA : http://en.wikipedia.org/wiki/National_Security_Agency
6. Site officiel de la NSA : http://www.nsa.gov/index.shtml
7. Article Wikipédia à propos de Bell LaPadula : http://en.wikipedia.org/wiki/Bell%E2%80%93LaPadula_model
8. Site officiel de SCC : http://www.securecomputing.com/index.cfm?sKey=738
9. Article Wikipédia à propos de MAC : http://en.wikipedia.org/wiki/Mandatory_access_control
10. Article Wikipédia à propos de SELinux : http://en.wikipedia.org/wiki/Security-Enhanced_Linux
11. Administration Linux Tome 3 - Chapitre 5-1
12. Article Wikipédia à propos des LSM : http://en.wikipedia.org/wiki/Linux_Security_Modules
13. Site officiel du projet Fedora : https://fedoraproject.org/fr/
14. Cycle de vie de Fedora : https://fedoraproject.org/wiki/Releases/Schedule
15. Site officiel des dépôts EPEL : https://fedoraproject.org/wiki/EPEL

semblable à RedHat sur certains paquets. Il est intéressant de noter que beaucoup de paquets non disponibles sur RedHat sont présents dans les dépôts EPEL qui sont par ailleurs documentés chez RedHat comme des dépôts à configurer [16].

Le choix aurait pu se porter sur une version avec une durées de vie plus longue, comme par exemple RedHat ou un de ces forks (voir boîtes informatives) (Centos [17], Scientific Linux [18], etc) qui proposent un support de 7 années. Ce choix est bien évidement pertinent mais varie selon les situations.

Fork :

Un fork, ou embranchement, est un nouveau logiciel créé à partir du code source d'un logiciel existant. Cela suppose que les droits accordés par les auteurs le permettent : ils doivent autoriser l'utilisation, la modification et la redistribution du code source. C'est pour cette raison que les forks se produisent facilement dans le domaine des logiciels libres. [19]

Dans les cas d'environnements de production où les mises à jour globales d'un système (upgrade) ne peuvent être réalisées toutes les deux versions, le choix d'une distribution Linux de type RedHat ou Centos est justifié. De plus, RedHat/Centos propose fréquemment (une fois par année en général) des versions intermédiaires de leur distribution (5.1, 5.2, 5.3, etc..) qui amènent nouveautés et patchs directement issus du développement de Fedora et de RedHat. Cependant, les ajouts de nouvelles technologies sont sélectionnés et il arrive fréquemment que des nouveautés soient reportées à une version majeur [20] suivante de Redhat. Ces reports ont pour raison principale la volonté de maintenir la stabilité de RedHat, même si pour cela des innovations mettent plus de temps pour parvenir à intégrer cette distribution Linux.

Pour comprendre comment choisir entre RedHat et Fedora et, dans quel contexte Fedora pourrait être un choix justifié, il est important de comprendre comment se déroule dans la pratique le développement croisé de RedHat et de Fedora. A un moment donné dans le cycle de sortie (tous les 6 mois, comme expliqué précédemment) de Fedora, RedHat pointe une version de Fedora et la retravaille pour la transformer en RedHat (la dernière étant Fedora 9 pour l'actuel RedHat Linux 5). Actuellement, Fedora 12 est utilisée pour créer RedHat Linux 6 tandis que Redhat Linux 6.1 apportera une partie des patchs et des fonctionnalités amenée par Fedora 13, 14 et 15.

Dans le cas où l'environnement est un espace de travail où les mises à jour sont possibles (production souple, laboratoires, postes de travails, etc), il est recommandé d'utiliser Fedora pour bénéficier des mises à jour de paquets, des patchs de sécurité dans leurs dernières versions, etc.

Dans le cas du laboratoire de HEPIA où les systèmes auront la possibilité d'être réinstallés très fréquemment, Fedora est un choix évidant. Le cas des serveurs de virtualisation de HEPIA pourra être traité de la même manière, mais avec une distribution au cycle de vie plus long (par exemple, Centos). Les explications qui vont suivre dans les sections suivantes de ce chapitre présenteront des techniques d'installation manuelle puis automatisées de Fedora, mais elles sont pleinement applicables au monde RedHat/Centos avec des modifications mineures.

7.2 Installation à partir du CD de NetInstall

Il est bien évidement tout à fait possible de créer un environnement de virtualisation sur son propre poste de travail ou sur un serveur à partir d'un simple CD d'installation de Fedora, Centos/RedHat ou toute autre distribution proposant un installateur évolué et souple. Les explications ci-dessous présentent les étapes pour, à partir d'un simple CD de NetInstall, créer un environnement de virtualisation complet.

16. Recherche du terme "EPEL" dans la documentation officiel de RedHat : https://www.redhat.com/search?q=EPEL&site=redhat_docs&filter=0
17. Site officiel de Centos : https://www.centos.org/
18. Site officiel de Scientific Linux : https://www.scientificlinux.org/
19. Article Wikipédia à propos de la notion de fort : http://en.wikipedia.org/wiki/Fork_(software_development)
20. Cycle de vie de RedHat : https://access.redhat.com/support/policy/updates/errata/

7.2.1 Télécharger et graver le CD d'installation

L'image du CD d'installation est disponible sur les différents miroirs de Fedora [21], il est cependant conseillé, depuis la Suisse, d'utiliser celui proposé gratuitement par `Switch` [22].

7.2.2 Installation à partir du "Setup"

Une fois la procédure d'installation débutée par Anaconda [23](l'installateur de Fedora, RedHat, Centos, etc.), suivez les instructions d'installation selon vos besoins. Le seul point important du Setup d'installation avant la sélection des paquets est de maintenir le choix par défaut d'utilisation de LVM(voir boîte d'information) au partitionnement.

`LVM (Logical Volume Manager) :`
La gestion par volumes logiques (en anglais, Logical Volume Management ou LVM) est une méthode et un logiciel de découpage, de concaténation et d'utilisation des espaces de stockage d'un serveur. Il permet de gérer, sécuriser et optimiser de manière souple les espaces de stockage en ligne dans les systèmes d'exploitation de type UNIX/Linux. [24] [25] [26] [27]

Une fois la phase de sélection des paquets arrivée, sélectionnez "customize now" et effectuez la sélection des paquets suivants (voir figure 7.1):

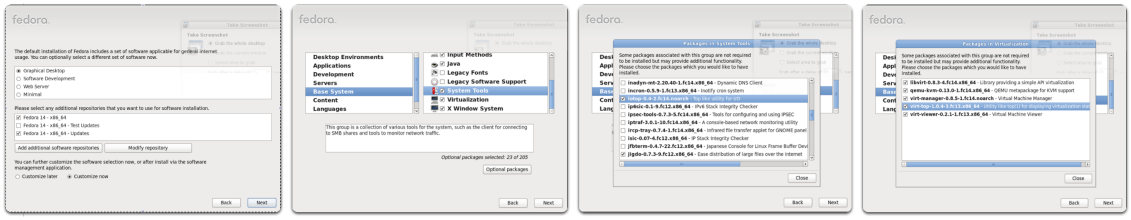


FIGURE 7.1: Capture de l'installation de Fedora et de la sélection des paquets

- Cocher le groupe "System Tools" puis, ajouter le logiciel "Iotop".
- Cocher le groupe "Virtualization" puis, ajouter le logiciel "Virt-top".

7.2.3 Installer les paquets supplémentaires

Une fois le système installé, il faut encore ajouter quelques outils utiles non sélectionables dans les paquets à l'installation. Les commandes suivantes (compatible Fedora/RedHat) présentent comment installer les paquets nécessaires.

```
yum install febootstrap febootstrap-supermin-helper
yum install guestfish libguestfs libguestfs-mount libguestfs-tools
yum install libguestfs-tools-c sysstat
```

L'utilité des différents outils, ainsi que leur choix, sera discuté dans les chapitres suivants selon leurs cas d'utilisation.

21. Liste de tous les miroirs de téléchargements Fedora : https://mirrors.fedoraproject.org/publiclist
22. CD de NetInstall sur le miroir Fedora de `Switch` : http://mirror.switch.ch/ftp/mirror/fedora/linux/releases/14/Fedora/x86_64/iso/Fedora-14-x86_64-netinst.iso
23. Documentation officiel de Anaconda : https://fedoraproject.org/wiki/Anaconda
24. Une introduction est disponible dans le livre Administration Linux Tome 1 - chapitre 6-5
25. Une explication avancées est disponible dans le livre Administration Linux Tome 2 - chapitre 8-1 à 8-34
26. Article d'introduction : http://www.markus-gattol.name/ws/lvm.html
27. Documentation de RedHat à propos de LVM : https://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/index.html , débute ici : https://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/ch-lvm.htm

7.2.4 Depuis un système déjà installé

Dans le cas où l'installation doit se faire sur un système Fedora ou autre installé, il faut installer les paquets et outils nécessaires à la main. Les commandes suivantes montrent comment réaliser cette installation sous Fedora.

```
yum install kvm
yum install virt-manager libvirt libvirt-python python-virtinst virsh
yum install febootstrap febootstrap-supermin-helper
yum install guestfish libguestfs libguestfs-mount libguestfs-tools
yum install libguestfs-tools-c sysstat
```

7.2.5 Obtenir les toutes dernière releases des logiciels de virttalisation

La documentation suivante[28] explique comment, en cas de besoin, obtenir les dernières version de la plupart des logiciels de virtualisation. Les logiciels disponibles dans ce dépôt sont souvent les versions qui viennent de sortir ou cours de phase finale de développement.

7.3 Installation à partir d'un fichier Kickstart

Ce chapitre présente comment à partir d'un fichier Kickstart[29] créer un environnement KVM basic sous Fedora.. Ce chapitre se présente sous la forme d'un tutoriel qui explique globalement la configuration d'un fichier kickstart et comment créer un CD de "Net-Install" guidé par un de ce fichier.

Pour composer un média d'installation personnalisé nous allons utiliser des fichiers Kickstart, très répandus dans le monde des distributions Linux basées sur `Red Hat`. Les fichiers Kickstart sont "consommés" par de nombreux logiciels, dont Anaconda[30] (l'installateur de Fedora) et les divers outils de composition comme Revisor[31] (Outil permettant de créer des medias d'installation ou des lives cd à partir d'un fichier Kickstart).

Ces logiciels n'utilisent pas forcement toutes les sections des fichiers Kickstart, Anaconda va utiliser le fichier Kickstart afin d'effectuer des installations automatiques, il aura donc besoin de toutes les informations disponibles. De son côté, Revisor par exemple ne s'intéresse qu'à la liste de paquets (ou groupe de paquets, précédés d'un @) que l'on désire inclure sur le média.

Il y a également une section %post, qui définit les actions à réaliser à la fin d'une installation, juste avant qu'Anaconda ne redémarrer l'ordinateur ainsi qu'une section %pre qui concerne les actions à réaliser avant l'installation.

L'exemple ci-dessous propose un fichier kickstart épuré qui permet de créer un système utilisant la langue anglaise, reposant sur une partition de type EXT3 et ayant mysql et mod auth mysql installé.

```
# Réponses utilisées par Anaconda
lang en_US.UTF-8
...
part /var --fstype ext3 --size=100 --grow
# Définition des paquets à installer/inclure
%packages
@mysql
...
mod_auth_mysql
# Actions à effectuer à la fin d'une installation
%post
```

28. Virtualization Preview Repository : https://fedoraproject.org/wiki/Virtualization_Preview_Repository
29. Documentation Fedora à propos des Kickstart : https://fedoraproject.org/wiki/Anaconda/Kickstart
30. Page principal de Anaconda : |https://fedoraproject.org/wiki/Anaconda
31. Page officiel de Revisor : http://revisor.fedoraunity.org/

Avant de songer à créer un média d'installation, il faut donc déterminer quels sont les éléments clés dont le fichier Kickstart sera responsable. Comme nous l'avons vu, celui-ci contient 4 principales parties à exploiter :

- Les réponses à donner à Anaconda afin d'automatiser le processus d'installation.
- La liste des logiciels à installer (packages).
- Que faire avant l'installation (pre).
- Que faire après que l'installation soit finie (post).

7.3.1 Environnement pour une workstation de virtualisation

A titre d'exemple, un simple fichier contenant tout ce qu'il est nécessaire pour installer un environnement KVM basé sur une distribution Fedora 14 est présenté ci-dessous. Pour des raisons de mise en page, certaines lignes ont été découpées.

```
# Kickstart my virutalisation workstation

#version=1.0

#This ks is used with an installation cd-rom
install
cdrom

#use english language for the system but fr_CH for the keyboard
lang en_US.UTF-8
keyboard fr_CH-latin1

#use DHCP to initialise default network settings
network --onboot yes --device eth0 --bootproto dhcp --noipv6

#define tiemzone
timezone --utc Europe/Zurich

#set root password, politic of passwords storage (SHA512+salt),
#enlable firewall and allowing ssh connection and enable selinux
#in enforcing mode.
rootpw  --iscrypted
$6$xlH68ru4FCpgq.bk$B6e585AFFnK
    MeV/sZVCHMLY3.rnJZoQFW.c.buMsQy3hVR.0FY2ldVpAr53YcG4ulKI5yeGAvWpfElVPHZ/AT1
selinux --enforcing
authconfig --enableshadow --passalgo=sha512 --enablefingerprint
firewall --service=ssh

# reset all the current disks
clearpart --all --drives=sda
# create a small part of 500Mb for /boot
part /boot --fstype=ext4 --size=500
# define à LVM physical volume of 500MB but able to grow if needed
part pv.XrW7Sr-7mmE-9TUq-OgB4-lA8b-UKFW-CGz8RR --grow --size=500

# create a group volume called vg_hacken on the created physical volume
volgroup vg_hacken --pesize=32768 pv.XrW7Sr-7mmE-9TUq-OgB4-lA8b-UKFW-CGz8RR
# Create a logical volume of 1024Mb with possiblity to grow to 51GB
logvol / --fstype=ext4 --name=lv_root --vgname=vg_hacken --grow --size=1024 --maxsize=51200
# Create a logical volume of 1008Mb with possibility to grow to 2GB
logvol swap --name=lv_swap --vgname=vg_hacken --grow --size=1008 --maxsize=2016
```

```
#Set the bootloader
bootloader --location=mbr --driveorder=sda --append="rhgb quiet"

#set repository from where the needed paquet will be found
repo --name="Fedora 14 - x86_64"
--baseurl=http://mirror.switch.ch/ftp/mirror/fedora/linux/releases/14/Everything/x86_64/os/
--cost=1000
repo --name="Fedora 14 - x86_64 - Updates"
--baseurl=http://mirror.switch.ch/ftp/mirror/fedora/linux/updates/14/x86_64/
--cost=1000

#set installation needed packages.
%packages
@admin-tools
@base
@core
@development-tools
@editors
@fedora-packager
@fonts
@gnome-desktop
@games
@graphical-internet
@graphics
@hardware-support
@input-methods
@java
@office
@online-docs
@printing
@sound-and-video
@system-tools
@text-internet
@virtualization
@base-x
xfsprogs
mtools
iscsi-initiator-utils
gpgme
bridge-utils
rpmdevtools
koji
mercurial
lua
rpmlint
plague-client
mock
bzr
openoffice.org-opensymbol-fonts
gvfs-obexftp
hdparm
gok
iok
vorbis-tools
jack-audio-connection-kit
gssdp
```

```
geoclue
createrepo
lzop
radeontool
enca
ebtables
iotop
festival
ntpdate
xsel
gupnp
fuse
PackageKit-command-not-found
ncftp
virt-top
sysstat
vim
gdm
febootstrap
febootstrap-supermin-helper
guestfish
libguestfs
libguestfs-mount
libguestfs-tools
libguestfs-tools-c
%end
```

Les différentes parties du fichiers ont été directement commentées dans le fichier présenté ci-dessus.

La dernière étape consiste à générer un CD d'installation qui utilise ce Kickstart. Ce CD sera réalisé à partir de l'image d'un CD de NetInstall de Fedora 14

La première étape consiste à récupérer une image d'un cd de netinstall.

```
[B@Hacken ~]$ wget http://mirror.switch.ch/ftp/mirror/fedora/linux/releases
/14/Fedora/x86_64/iso/Fedora-14-x86_64-netinst.iso
--2011-04-04 16:39:56--  http://mirror.switch.ch/ftp/mirror/fedora/linux/releases/
14/Fedora/x86_64/iso/Fedora-14-x86_64-netinst.iso
Resolving mirror.switch.ch... 130.59.10.36, 2001:620:0:8::20
Connecting to mirror.switch.ch|130.59.10.36|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 230686720 (220M) [application/octet-stream]
Saving to: ‘‘Fedora-14-x86_64-netinst.iso’’

100%[======================================
====================================>] 230,686,720 2.79M/s   in 84s

2011-04-04 16:41:20 (2.62 MB/s) - ‘‘Fedora-14-x86_64-netinst.iso’’ saved [230686720/230686720]
```

Puis, en extraire le contenu dans un répertoire

```
[B@Hacken ~]$ mkdir fedora_virt_gui_netinstall
[B@Hacken ~]$ su -
Password:
[root@Hacken ~]# cd /home/B/
[root@hacken B]# mkdir netinstallcd
[root@Hacken B]# mount -o loop Fedora-14-x86_64-netinst.iso netinstallcd/
mount: warning: netinstallcd/ seems to be mounted read-only.
```

```
[root@Hacken B]# cp -r netinstallcd/* fedora_virt_gui_netinstall/
[root@hacken B]# mkdir fedora_virt_gui_netinstall/isolinux/ks
[root@Hacken B]# ll fedora_virt_gui_netinstall/
total 12
dr-xr-xr-x 3 root root 4096 Apr  4 16:45 EFI
drwxr-xr-x 3 root root 4096 Apr  4 16:45 images
drwxr-xr-x 2 root root 4096 Apr  4 16:45 isolinux
[root@Hacken B]# umount netinstallcd/
[root@Hacken B]# chown -R B.users fedora_virt_gui_netinstall/
```

Définir le boot sur le fichier kickstart et créer ce dernier.

```
[B@Hacken ~]$ vim fedora_virt_gui_netinstall/isolinux/isolinux.cfg
.
.
label Fedora_Kick_Screen
  menu label Install with kickstart and auto_screenshot
  menu default
  kernel vmlinuz
  append initrd=initrd.img ks=cdrom:/isolinux/ks/fedora_virt_gui_netinstall.ks
.
.
[root@hacken B]# vim fedora_virt_gui_netinstall/isolinux/ks/fedora_virt_gui_netinstall.ks
```

Finalement, il ne reste plus qu'a créer le CD bootable pour l'installation.

```
B@Hacken fedora_virt_gui_netinstall]$ su -
Password:
[root@Hacken ~]# cd /home/B/fedora_virt_gui_netinstall/
[root@Hacken fedora_virt_gui_netinstall]# mkisofs -o Fedora-Kickstart-screen.iso
-b isolinux/isolinux.bin -c isolinux/boot.cat -no-emul-boot -boot-load-size 4 -boot-info-table -R -J -v
I: -input-charset not specified, using utf-8 (detected in locale settings)
genisoimage 1.1.11 (Linux)
Scanning .
Scanning ./images
Excluded: ./images/TRANS.TBL
Scanning ./images/pxeboot
Excluded: ./images/pxeboot/TRANS.TBL
Scanning ./EFI
Scanning ./EFI/BOOT
Excluded: ./EFI/BOOT/TRANS.TBL
Scanning ./isolinux
Excluded: ./isolinux/TRANS.TBL
Scanning ./isolinux/ks
Excluded by match: ./isolinux/boot.cat
Writing:   Initial Padblock                     Start Block 0
Done with: Initial Padblock                     Block(s)    16
Writing:   Primary Volume Descriptor            Start Block 16
Done with: Primary Volume Descriptor            Block(s)    1
Writing:   Eltorito Volume Descriptor           Start Block 17
Size of boot image is 4 sectors -> No emulation
Done with: Eltorito Volume Descriptor           Block(s)    1
Writing:   Joliet Volume Descriptor             Start Block 18
Done with: Joliet Volume Descriptor             Block(s)    1
Writing:   End Volume Descriptor                Start Block 19
Done with: End Volume Descriptor                Block(s)    1
Writing:   Version block                        Start Block 20
Done with: Version block                        Block(s)    1
```

```
Writing:   Path table                              Start Block 21
Done with: Path table                              Block(s)    4
Writing:   Joliet path table                       Start Block 25
Done with: Joliet path table                       Block(s)    4
Writing:   Directory tree                          Start Block 29
Done with: Directory tree                          Block(s)    7
Writing:   Joliet directory tree                   Start Block 36
Done with: Joliet directory tree                   Block(s)    7
Writing:   Directory tree cleanup                  Start Block 43
Done with: Directory tree cleanup                  Block(s)    0
Writing:   Extension record                        Start Block 43
Done with: Extension record                        Block(s)    1
Writing:   The File(s)                             Start Block 44
  3.86% done, estimate finish Mon Apr  4 17:08:17 2011
  7.71% done, estimate finish Mon Apr  4 17:08:17 2011
 11.57% done, estimate finish Mon Apr  4 17:08:17 2011
 15.43% done, estimate finish Mon Apr  4 17:08:23 2011
 19.27% done, estimate finish Mon Apr  4 17:08:22 2011
 23.13% done, estimate finish Mon Apr  4 17:08:21 2011
 26.98% done, estimate finish Mon Apr  4 17:08:24 2011
 30.84% done, estimate finish Mon Apr  4 17:08:23 2011
 34.69% done, estimate finish Mon Apr  4 17:08:22 2011
 38.55% done, estimate finish Mon Apr  4 17:08:22 2011
 42.40% done, estimate finish Mon Apr  4 17:08:21 2011
 46.26% done, estimate finish Mon Apr  4 17:08:23 2011
 50.11% done, estimate finish Mon Apr  4 17:08:22 2011
 53.97% done, estimate finish Mon Apr  4 17:08:22 2011
 57.82% done, estimate finish Mon Apr  4 17:08:22 2011
 61.68% done, estimate finish Mon Apr  4 17:08:21 2011
 65.53% done, estimate finish Mon Apr  4 17:08:21 2011
 69.39% done, estimate finish Mon Apr  4 17:08:22 2011
 73.25% done, estimate finish Mon Apr  4 17:08:23 2011
 77.10% done, estimate finish Mon Apr  4 17:08:23 2011
 80.96% done, estimate finish Mon Apr  4 17:08:24 2011
 84.81% done, estimate finish Mon Apr  4 17:08:26 2011
 88.67% done, estimate finish Mon Apr  4 17:08:27 2011
 92.52% done, estimate finish Mon Apr  4 17:08:27 2011
 96.38% done, estimate finish Mon Apr  4 17:08:27 2011
Total translation table size: 8056
Total rockridge attributes bytes: 3768
Total directory bytes: 13154
Path table size(bytes): 90
Done with: The File(s)                             Block(s)    129517
Writing:   Ending Padblock                         Start Block 129561
Done with: Ending Padblock                         Block(s)    150
Max brk space used 22000
129711 extents written (253 MB)
```

7.4 Spécifications de l'environnement du laboratoire de virtualisation HEPIA

Afin de remplacer les laboratoires de virtualisation ESX actuellement utilisé à HEPIA, une infrastructure d'installation basée sur l'utilisation d'un serveur PXE doit être réalisée. L'utilisation de PXE est un choix réalisé par le laboratoire pour maintenir l'utilisation de technologies connues.

Comme le temps à disposition pour ce travail est limité, ce document décrit les spécifications de ce qu'il faudra faire, mais les tâches de tests, de réalisation et de création des spécifications finales seront réalisées par les assistants du laboratoire.

7.4.1 Configurations machines

Les machines à disposition proposent la configuration suivante :

- Processeur Intel double coeur 2.4Ghz avec extension VT.
- 4GB de mémoire vive.
- 1 disque dur de 300GB.
- Trois interfaces réseaux.

Chaque place de travail du laboratoire disposera de deux machines avec des installations différentes.

- Une installation avec interface graphique, un environnement de virtualisation KVM et des outils de développement.
- Une installation sans environnement graphique avec un environnement virtualisation KVM.

7.4.2 Configuration disque

En fonction de la configuration, un partitionnement LVM dédié doit être réalisé. Trouver le bon partitionnement adapté aux besoins du laboratoire est une chose complexe. Cette tâche a été réalisée par Cédric Penas (assistant au laboratoire de virtualisation de l'HEPIA) à partir d'une démarche rédigée dans ce document. La présente section reporte cette démarche dans ce rapport.

1 : Documentation

Se documenter sur LVM [32] [33] [34] [35] pour prendre conscience de ses fonctionnalités.

Il peut aussi être intéressant de prendre connaissance des bonnes pratiques [36] IBM pour KVM. Ces bonnes pratiques datent un peu mais, il est important de les prendre en considération sans forcement les appliquer à la lettre.

2 : Cas pratiques.

Les serveurs utilisés dans le cadre de ces tests ont un seul disque dur. De ce fait, les considérations telles que la répartition des IO ou la redondance/raid ne sont pas applicables dans le cas présent.

De manière générale, pour un simple système de laboratoire avec un seul disque, le découpage minimal suivant est conseillé : (Pensé pour un stockage de 320Gb disponible)

- 1 - Une partition pour le système de boot (/boot) (1 GB).
- 2 - Un système logique LVM sur le reste du système et les VMs.

32. Petite introduction à LVM : http://www.markus-gattol.name/ws/lvm.html
33. Documentation RedHat : https://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/index.html Débute ici : |https://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/ch-lvm.html
34. Introduction dans le livre "Linux Administration Tome 1" : Chapitre 6-5 : LVM- les éléments clés
35. Explications complète dans le livre "Linux Administration Tome 2" : Chapitre 8-1 à 8-34 LVM
36. Bonnes pratiques IBM pour KVM : http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaat/liaatbestpractices_pdf.pdf

- 2 - 1 - Un volume logique pour le system (/) (25-+ GB).
- 2 - 3 - Un volume logique pour (/var/log) (10 GB).
- 2 - 4 - Un volume logique pour la swap (2x la RAM).
- 3 - Le reste libre
- 3- 1 - Augmenter la taille de la partie système si beaucoup de machines virtuelles doivent être hébergées sous la forme de fichiers

En cas de création d'un système virtualisé il faut :

- 1- Créer un volume logique pour stocker la VM.
- 2 - Créer le mapage libvirt vers un bloc device.

Voici un exemple de mappage entre un volume logique et un "bloc device" :

```
<disk type='block' device='disk'>
<driver name='qemu' type='raw' cache='writeback'/>
<source dev='/dev/lvm-raid/vtest2-hdd0'/>
<target dev='vda' bus='virtio'/>
<alias name='virtio-disk0'/>
<address type='pci' domain='0x0000'
bus='0x00' slot='0x04' function='0x0'/>
</disk>
```

Il est important d'utiliser des devices virtio pour utiliser la paravirutalisation (Modèle bloc à bloc).

- 3 - Créer une machine avec le device virtuel comme disque.
- 4 - Démarrer la machine.

Afin d'obtenir une meilleure expertise dans l'utilisation de LVM, il est possible de réaliser les manipulations suivantes :

- Création d'un volume logique.
- Extension d'un volume logique.
- Ajout d'un disque à un VG.
- Ajout d'un LV sur un VG.
- Créer un snapshot d'un volume logique contenant une machine virtuelle.
- Le restaurer

Cette démarche doit permettre d'attendre les objectifs minimaux pour être capable d'administrer de manière efficace un environnement KVM.

- 1 – S'habituer au fonctionnement de LVM, déterminer et documenter les manipulations importantes pour le contexte.
- 1 – Création de machines.
- 1 – Clonage de machine vers un autre système logique LVM.
- 1 – Redimensionnement des volumes logiques d'une machine virtuelle déjà crée (LVM sur l'hyperviseur et LVM dans la VM).

Ces manipulations seront réalisées par Cédric Penas assistant dans le laboratoire de virtualisation de HEPIA.

7.4.3 Configuration du laboratoire

Dans le cadre du laboratoire de virtualisation de l'HEPIA, l'architecture suivante devrait être mise en œuvre. Ce plan a été construit à partir de discussions avec les mandants du projets et les assistants du laboratoire de virtualisation. Description générique des différents éléments :

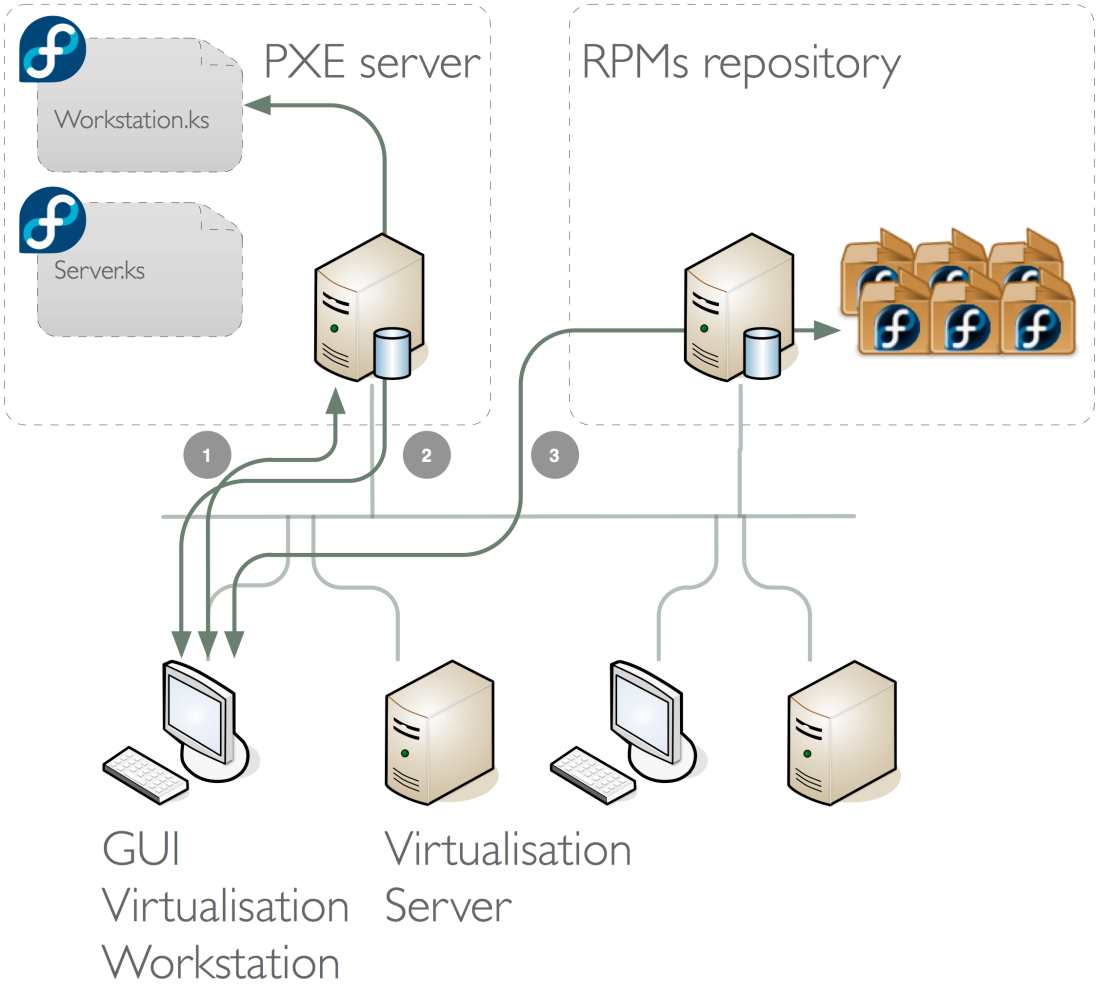


FIGURE 7.2: Système d'installation automatisé basé sur un serveur PXE

- Un serveur PXE contenant une image minimum pour démarrer un système d'installation.
- Des fichiers Kickstart dont la sélection se fera via le système minimum d'installation disponible sur le serveur PXE.
- Un dépôt de paquets rpm pour que le processus d'installation puisse récupérer les paquets nécessaires.

PXE (Pre-boot eXecution Environment) :

Le démarrage PXE permet à une station de travail de démarrer depuis le réseau en récupérant une image de système d'exploitation qui se trouve sur un serveur. L'image ainsi récupérée peut être le système d'exploitation brut ou bien le système d'exploitation personnalisé avec des composantes logicielles (suite bureautique, utilitaires, packs de sécurité, scripts, etc...). Une fois cette image "pré-chargée", elle peut éventuellement, en fonction des paramétrages passés à cette image sur le serveur, être installée sur la machine qui a booté en PXE. [37]

37. Article Wikipédia à propos de PXE : http://en.wikipedia.org/wiki/Preboot_Execution_Environment

Globalement, le processus d'installation sera le suivant (exemple pour la workstation de virtualisation) :

- 1 : La machine démarre en PXE et charge l'image de base proposée par le serveur.
- 2 : Le système contenu dans l'image propose de démarrer un processus d'installation à partir de l'un des fichiers kickstart disponibles.
- 3 : Une fois le processus d'installation démarré avec le bon fichier kickstart, le processus d'installation installe le système avec les paquets disponibles dans le dépôt (le dépôt est celui que le fichier kickstart demande d'utiliser).

Cette architecture peut, en plus de permettre d'installer les "workstation" et les serveurs de virtualisation, proposer des fichiers kickstarts dédiés à des machines virtuelles.

Approche manuelle

L'approche manuelle de réalisation d'un tel dispositif demande de choisir et configurer tous les éléments à la main.

Serveur ou logiciel capable de fournir un menu de boot ou une image système de base.
Ici, un logiciel comme pxelinux[38] de syslinux[39] permet de réaliser ceci de manière assez simple[40] (suivant la qualité de l'interface graphique voulue).

De manière générale, pxelinux va au moment du boot pxe, à l'aide d'un fichier de configuration, sélectionner quel système (image minimal déclenchant le mécanisme d'installation) charger.

Dépôt de fichiers :
Créer un dépôt de fichier rpm est extrêmement simple. L'utilitaire createrepo[41] fait pratiquement tout tout seul[42]. Il est aussi possible de créer des politiques de caches[43] en cas de besoins particuliers.

Approche automatique

Une autre technique pour la création d'un tel environnement est d'utiliser un produit qui le fera (hors dépôts rpm). Le produit conseillé dans ce domaine est cobbler[44]. Cobbler est en fait un centre d'installation et de création de système basés rpm. Il prend en compte le boot pxe pour l'installation de serveur physique mais, permet aussi la création de machines virtuelles, de live-cd, de CD d'installation, etc.

La présentation suivante[45] présente de manière superficielle les fonctionnalités de base de Cobbler. Dans un environnement virtuel ou non, cobbler est un outil extrêmement puissant pour la création et la mise à jour de systèmes physiques et virtuels.

Cette méthode est évidemment l'approche conseillée pour la problématique présente dans ce travail et pour la gestion de parc de machines virtuelles et physiques. Le temps à disposition pour ce travail ne permettant pas de monter un système d'exemple, cette tâche sera réalisée par les assistants du laboratoire de virtualisation de l'HEPIA.

Approche par live-cd

Une autre approche est aussi l'utilisation de live-cd. L'outil livecd-tool permet la création de live-cd[46].

38. pxelinux chez syslinux : http://syslinux.zytor.com/wiki/index.php/PXELINUX
39. Site officiel de syslinux : http://syslinux.zytor.com/wiki/index.php/The_Syslinux_Project
40. Exemple de création d'un serveur pxelinux très basique : http://doc.fedora-fr.org/wiki/Configuration_d'un_serveur_pour_lancer_des_installations_par_PXE_boot
41. Page officiel de createrepo : http://createrepo.baseurl.org/
42. Exemple de documentation sur la création de dépôt Fedora : https://fedoraproject.org/wiki/Extras/CreateRepo
43. Politique de cache de YUM : http://yum.baseurl.org/wiki/YumMultipleMachineCaching
44. Site officiel de cobbler : https://fedorahosted.org/cobbler/
45. Introduction à cobbler : http://mdehaan.fedorapeople.org/presentations/cobbler/index.html#(1)
46. Documentation détaillée de Revisor : https://www.ohloh.net/p/livecd-tools

Bien que livecd-tool permet une certaine simplicité dans la création de live-cd[47], beaucoup de problèmes d'intégrations peuvent survenir. Créer un live-cd demande donc une certaine quantité de travail et de tests. Cependant, c'est une solution élégante pour créer un environnement de virtualisation temporaire de manière rapide et simple.

Les principales étapes suivantes sont nécessaires pour créer un tel CD :

- 1 - Déterminer les paquets et outils nécessaires (Étape normalement réalisée dans la phase de création du Kickstart) et en faire un fichier Kickstart.
- 2 - Utiliser Revisor pour créer le media live.
- 3 - Tester le media live.
- 4 - Corriger les problèmes qui pourraient survenir à l'aide des zones de scriptes disponibles dans le fichier Kickstart.

7.5 Rappel des divers solutions

Le tableau si-dessous présente un rappels des différentes solutions proposées.

Solution	Avec dépôt rpm local	Sans dépôt rpm local	Considérations	Considérations logistique
DVD de base	-	-	- Les paquets doivent être sélectionnés à la main	+ Demande le moins de logistique
CD de Netinstall de base	+ Installation plus rapide	- Installation plus lente	- Les paquets doivent être sélectionnés à la main	+ Demande le moins de logistique suivant la présence ou non d'un dépôt local
DVD guidé par kickstart	-	-	+ Sélection automatique des paquets	+ Demande pas beaucoup de logistique
CD de Netinstall guidé par Kickstart	+ Installation plus rapide	- Installation plus lente	+ Sélection automatique des paquets	+ Demande pas beaucoup de logistique suivant la présence ou non d'un dépôt local
Netinstall PXE guidé par Kickstart	+ Installation plus rapide	- Installation plus lente	+ Sélection automatique des paquets	+ Logistique moyenne (PXE + dépôt su présent)
			+ Plusieurs installations possible depuis un même PXE	
Netinstall PXE guidé par Kickstart (Cobbler)	+ Installation plus rapide	- Installation plus lente	+ Sélection automatique des paquets	+ Logistique moyenne (Cobbler + dépôt su présent)
			+ Plusieurs installations possible depuis un même PXE	- Demande du temps pour configurer et apprendre à utiliser
			+ Possible installations de VM via PXE ou en statiques	
			+ Beaucoup de possibilités	- demande du temps pour créer
Live-CD	-	-	+ Solution la plus souple	+ Demande le moins de logistique
			- Les environnements crées sont temporaires	- demande du temps pour créer

FIGURE 7.3: Tableau de résumé des solutions d'installations

7.6 Conclusion

Ce chapitre a permis de voir les différentes étapes permettant de créer un système avec un environnement de virtualisation KVM et d'en automatiser la procédure. L'utilisation de fichiers Kickstart permet la personnalisation fine de l'installation d'un système tandis que l'utilisation d'outils comme cobbler permet le déploiement rapide et automatisé de systèmes variés.

Les activités qui seront réalisées par les assistants du laboratoire de virtualisation permettront de définir précisément les spécifications de ces systèmes et de créer un environnement de virtualisation complet ou l'installation des stations de virtualisation ainsi que des machines virtuelles seront totalement automatisées et guidées.

47. Documentation avec exemples sur la création de live-cd Fedora : https://fedoraproject.org/wiki/FedoraLiveCD/LiveCDHowTo

Le chapitre suivant s'intéresse au moyen de créations de machines virtuelles et à l'automatisation de ces créations.

Chapitre

8

Création de machines virtuelles

Sommaire

vir bonus est is, qui prodest quibus potest, nocet nemini

Cicéron, De Officiis, 3.64.

La mise en place d'un environnement de virtualisation complet passe par la création du système d'hébergement, la création de machines virtuelles et leurs maintenances. Ce chapitre s'intéresse à la seconde étape. Le chapitre précédent a permis de créer et d'automatiser la création des serveurs d'hébergement. Ce chapitre présentera quelques méthodes pour créer des machines virtuelles.

La première partie du chapitre s'intéressera à la création manuelle et semi-automatique de machines virtuelles tandis que la seconde partie présentera une solution novatrice pour créer une fabrique et un système de déploiement automatique de machines virtuelles.

8.1 Outils pour la création de machines virtuelles à la main

Cette section présente quelques (n'a pas vocation d'être exhaustif) outils pour créer des machines virtuelles utilisables sous KVM.

8.1.1 qemu

Les outils disponibles de base avec KVM permettent de créer relativement simplement des machines virtuelles.

Créer une image pour accueillir la future machine virtuelle.

```
qemu-img create -f qcow2 <Image_Name> <size>
```

Installer un système dedans (Ici depuis un cd-rom).

```
qemu-kvm -hda <Image_Name> -m 512
-cdrom </Path/to/the/ISO/Image> -boot d -vga std
```

Cette procédure simplifiée permet la création d'une machine de base et son installation. Mais, comme il a été vu précédemment, la commande qemu-kvm se révèle souvent extrêmement fastidieuse à utiliser. De plus, les systèmes créés de cette manière ne seront pas visibles sous virt-manager ou tout système reposant sur libvirt. Pour les rendre visibles avec ces systèmes, il est nécessaire de l'importer dans libvirt.

```
virt-install --import --accelerate --disk path=<Image_Name>
--os-type linux --os-variant <OS> --ram 512 --name <New Name>
```

Lorsque un système est importé dans un environnement libvirt, le hardware virtualisé doit-être redéfini.

8.1.2 Création avec libvirt

libvirt fournit de base toute une panoplie d'outils pour créer et administrer un système de virtualisation. Il ne sera présenté ici que ceux pour créer des machines virtuelles.

La commande suivante crée une machine virtuelle, en définit le hardware et en prépare l'installation via internet en pointant directement sur une image système minimal déclenchant une installation via internet.

```
virt-install --ram=512 --vcpus=2
 --vnc --network=bridge:virbr0 --keymap=fr-ch --os-type=linux --os-variant=fedora14
--name=test2 --file=/var/lib/libvirt/images/test2.img --file-size=8
--location=http://mirror.switch.ch/ftp/mirror/fedora/linux/releases/14/Fedora/x86_64/os/
```

Descriptions des paramètres utilisés :

- –ram : quantité de mémoire allouée.
- –vcpus : quantité de processeurs virtuels.
- –vnc : créer un serveur vnc.
- –network : paramètres réseaux. dans l'exemple présenté, se branche sur le bridge virtuel virb0.
- –keymap : type de clavier.
- –is-type et –os-variant: informations complémentaires sur le système virtualisé.
- –name : nom avec lequel la machine est visible dans libvirt.
- –file : emplacement de la machine virtuelle
- –file-size : taille de l'image disque (ici : 8GB).
- – location : emplacement du périphérique ou de l'emplacement à démarrer.

Cette commande propose beaucoup d'arguments et la lecture des pages de manuel man virt-install répond à beaucoup de questions possibles sur l'utilisation de cette commande.

Chaque machine virtuelle est stockée avec des informations XML qu'il est possible de visualiser.

```
virsh dumpxml  <Machine name>
```

Par exemple, ci-dessous : une machine virtuelle nommée "test".

```
<domain type='kvm'>
  <name>test</name>
  <uuid>e9b4c7be-d60a-c16e-92c3-166421b4daca</uuid>
  <memory>524288</memory>
  <currentMemory>524288</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='x86_64' machine='pc-0.13'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw'/>
      <source file='/var/lib/libvirt/images/test.img'/>
      <target dev='vda' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </disk>
    <disk type='block' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <target dev='hdc' bus='ide'/>
      <readonly/>
      <address type='drive' controller='0' bus='1' unit='0'/>
    </disk>
    <controller type='ide' index='0'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
    </controller>
    <interface type='network'>
      <mac address='52:54:00:cc:1c:10'/>
      <source network='default'/>
      <target dev='vnet0'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
```

```
    </console>
    <input type='tablet' bus='usb'/>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='-1' autoport='yes'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </sound>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </video>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

Il est aussi possible de créer une machine virtuelle directement depuis un fichier XML de description.

```
virsh create machine.xml
```

La création à la main de machines virtuelles est donc une étape plutôt longue qui, avec libvirt, peut être simplifiée par la création de fichiers XML dédies.

Il est cependant plus facile d'utiliser l'interface graphique virt-manager pour générer ces fichiers xml et manipuler ces machines virtuelles avec libvirt (au travers de l'interface graphique de virt-manager).

8.1.3 Virt-Manager

Le bouton Nouveau permet de lancer l'assistant de création.

- L'étape 1 permet de choisir le nom de la nouvelle machine et le mode d'installation. Il est possible d'installer depuis un lecteur de DVD, une image ISO enregistrée sur le disque local (notre exemple), depuis le réseau ou en mode PXE.
- L'étape 2 permet de définir le lecteur ou l'image à utiliser. On définira aussi le système cible à partir d'une liste très complète. Ce choix permettra à l'application de proposer des réglages prédéfinis et adaptés dans la suite.
- L'étape 3 permet de définir les ressources allouées à la machine virtuelle
- L'étape 4 permet de définir la taille de l'image disque à créer (il est aussi possible de réutiliser une image existante). L'option "Allocate entire disk now" permet d'améliorer les performances de la machine virtuelle en allouant immédiatement la taille spécifiée. Ne pas l'utiliser permet de réduire de manière significative l'espace disque.

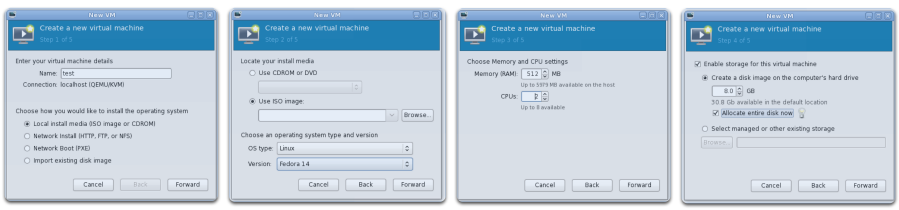


FIGURE 8.1: Étape 1 à 4

L'étape 5 (8.2) permet de contrôler les informations collectées dans les étapes précédentes et de confirmer la création et le lancement de la machine virtuelle. Les options avancées permettent de définir le mode de

FIGURE 8.2: Étape 5

connexion au réseau. Le bouton Terminer permet de lancer la création de la machine et lance son démarrage à partir du support choisi à l'étape 2.

On obtient une console (8.3) sur la machine virtuelle et on peut commencer l'installation du système. Après l'installation du système et son redémarrage, on obtient une machine virtuelle opérationnelle.

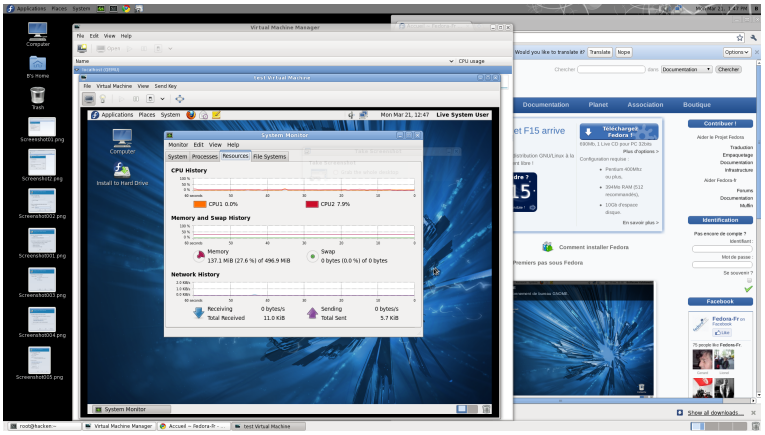


FIGURE 8.3: Machine virtuelle en fonction

virt-manager permet de manière très simple de gérer des manipuler des machines virutel en local ou sur un système distant. Cependant, il n'est pas nécessaire dans le cadre de ce travail de s'intéresser à son fonctionnement détaillé.

8.2 Créer une fabrique de machine virtuelles avec des outils open-source

Cette section s'intéresse à l'outil BoxGrinder [1] qui permet de créer un système de création et de déploiement de machines virtuelles. Le but ici n'est pas de d'écrire précisément son fonctionnement mais, de donner un aperçu global pour que l'étude de ce projet puisse être débutée par les assistants du laboratoire de virtualisation de l'HEPIA. Cet outil doit permettre dans le cadre de différents projets de créer un mécanisme de fabrication de machines virtuelles dédiées à partir d'une même base.

1. Site officiel de BoxGrinder : http://boxgrinder.org/

La figure 8.4 présente un système boxgrinder. Un système boxgrinder est composé de trois types de modules

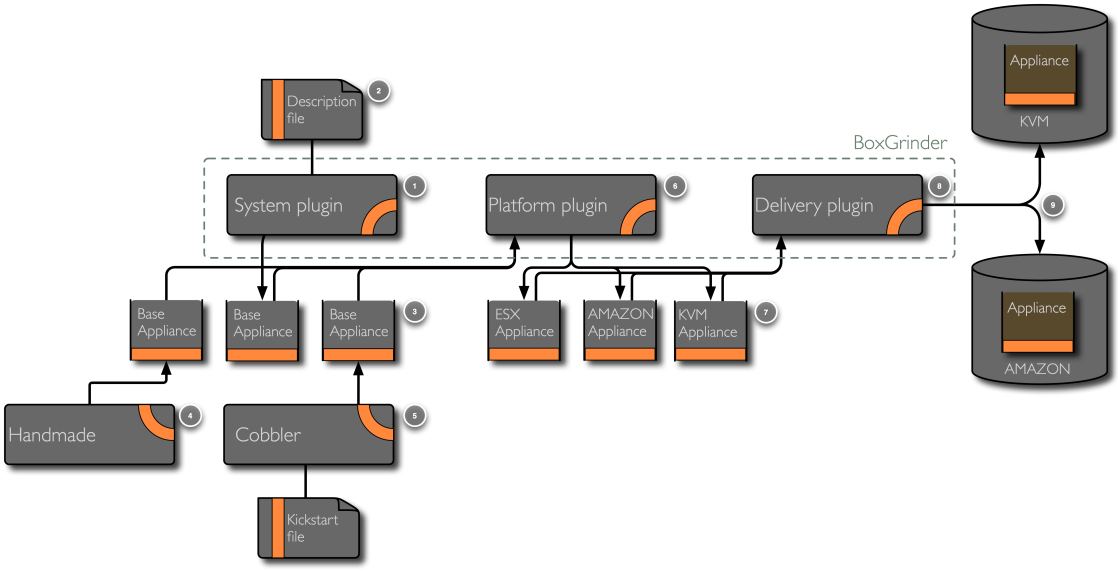


FIGURE 8.4: Système de création automatisé de machines virtuelles

(appelés plugins dans la terminologie de boxgrinder) :

- System plugin (1) : permet la création de machines virtuelles (3) en fonction d'un fichier de description (2).
- Platform plugin (6) : permet la conversion et la personnalisation de machines virtuelles (3) provenant du plugin system (1), de machines réalisées à la main (4) ou via un outil de génération comme coobler (5). Ces machines sont converties vers le format de l'environnement où elles devront être utilisées (7).
- Delivery plugins (8) : permet de déployer les machines virtuelles (7) dans leurs environnements de production (9).

Un tel outil permet donc, à partir de modèle de machine virtuelle, de créer des "appliances" spécialisées (logiciels, configurations, binaires, etc) et de les déployer de manière efficiente vers leurs environnements de production.

8.3 Conclusion

Ce chapitre a permis de passer brièvement en revue différents moyens de créer des machines virtuelles pour KVM. Les outils manuels ne sont clairement pas compatible avec une utilisation en masse et les outils proposés par libvirt répondent à ce problème.

Les solutions graphiques telles que virt-manager permettent d'utiliser les outils proposés par libirt de manière simple et donne des moyens basiques de monitoring.

Finalement, les outils comme boxgrinder permettent d'automatiser la création et la personnalisation de machines virtuelles à partir de modèles et font partie des éléments à mettre en oeuvre pour la création d'un environnement complet et automatisé de virtualisation.

Ce chapitre termine cette introduction au monde de KVM, les prochains chapitres reviennent sur le déroulement du projet, les objectifs accomplis et conclut ce travail d'approfondissement.

IV

Conclusions
Constatations

Sommaire

Chapitre

9

Suivi de projet

Le mieux est l'ennemi du bien.
Voltaire - Dictionnaire Philosophique (1764).

9.1 Planification et suivi

Le tableau ci-dessous résume les différentes étapes de ce projet. Chaque point ainsi que les difficultés rencontrées sont discutées dans les sections suivantes.

De par la nature de veille de ce travail, sa planification a été volontairement simplifiée (en commun accord entre l'étudiant et le professeur responsable du projet). De cette manière, il est plus facile de faire évoluer le cahier des charges du projet au cours du temps, tout en maintenant une structure et un suivi.

Tâches	**Temps prévu**	**Temps effectué**
Analyses et critiques de l'architecture de KVM et des outils utiles	80 heures	150 heures
Mise en pratique de KVM	40 heures	60 heures
Analyse et critiques de l'architecture de SheepDog	30 heures	10 heures
Rapport (hors parties principales)	30 heures	60 heures
Temps total	180 heures	290 heures

Le tableau montre un dépassement de temps clairement visible. Les sections suivantes expliqueront où ont été rencontrées les difficultés.

A l'origine de ce projet, l'effort nécessaire pour remettre à jour les compétences techniques au sujet du noyau Linux de l'auteur de ce projet ont mal été estimées. Cette erreur d'estimation a causé une grosse consommation de temps au début du projet. La consommation en termes de rappels techniques est estimée à environ 70 heures (relecture de chapitres dans divers livres, lecture de codes sources, réalisations de petits tests, etc). Le second gros point consommateur aura été l'étude de KVM et des outils connexes (vitrio, libvirt, etc.) qui comporte un temps d'analyse de code réellement conséquent (plusieurs 10ènes de milliers de lignes de codes).

9.2 Analyses et critiques de l'architecture de KVM et des outils utiles

9.2.1 Architecture de KVM

- Rappels sur l'architecture du noyau Linux.
- Recherche de documentation sur KVM.
- Étude du code source de KVM.
- Réaliser des tests à l'aide de KVM et des outils de développement à disposition pour bien comprendre le sujet.
- Documentation des recherches effectuées dans le rapport.

Ce qui a été réalisé

- Étude de KVM.
- Écrire la documentation sur KVM et son architecture.

9.2.2 Outils utiles

- Rechercher et sélectionner les principaux outils à utiliser pour gérer de manière efficiente un environnement KVM au quotidien.
- Comprendre les différents outils.
- Écrire de la documentation sur l'architecture de ces outils.

Ce qui a été réalisé

- Étudier, analyser et documenter l'architecture des drivers paravirtualisés virtio.
- Étudier, analyser et documenter l'architecture de la librairie libvirt.
- Étudier le fonctionnement d'outils fonctionnant avec libvirt (Par exemple : virt-clone, v2v, virt-top, vrish, etc.).
- Étudier le fonctionnement de KSM appliqué à KVM.
- Effectuer des tests de MOM (memory overcommitement manager) [1] [2].
- Étudier le fonctionnement des librairies libguestfs [3]

Ce qui n'a pas été réalisés

- Analyse et documentation des outils basés sur libvirt.
- Documenter le fonctionnement de KSM avec KVM.
- Documentation sur MOM.
- Analyse et documentation des librairies libguestfs.

1. Page officiel de MOM : https://github.com/aglitke/mom/wiki/
2. Documentation IBM sur MOM et la problématique de l'overcommitement sous KVM: http://www.ibm.com/developerworks/linux/library/l-overcommit-kvm-resources/index.html
3. Site officiel de libguestfs : http://libguestfs.org/

9.2.3 Difficultés rencontrées

De manière générale, les difficultés sont les suivantes (classifiées par ordre décroissant de consommation de temps) :

1. Temps nécessaire pour effectuer les rappels techniques nécessaires.
2. Temps nécessaire pour écrire la documentation.
3. Compréhension de la partie paravirtualisation de KVM.
4. Sélectionner le bon outil pour les bonnes tâches de maintenance de KVM.
5. Temps nécessaire à certains tests qui peuvent être très long (ex : développement des fichier kickstart et tests de ceux-ci, multiples créations de machines virtuelles, tests de consommations mémoires, etc).

9.3 Mise en pratique de KVM

Cette partie vise, par l'intermédiaire de cas tests, à montrer l'utilisation de KVM et la résolution de problèmes sélectionnés.

- Effectuer des tests pour obtenir une meilleure expertise sur l'environnement KVM.
- Créer des cas tests.
- Réaliser les cas tests.
- Documenter les cas réalisés.

9.3.1 Ce qui a été réalisé

- Tester et documenter l'installation d'un environnement KVM à la main.
- Tester et documenter comment automatiser la création d'un environnement KVM complet (Installation automatique, distributions de l'installation, gestions des machines, snapshot, etc)
- Tester et documenter comment créer des machines virtuelles.
- Tester et documenter comment automatiser la création de machines virtuelles.

9.3.2 Ce qui n'a pas pu être réalisé

- Documenter l'utilisation d'outils pour assurer le monitoring d'un parc de machines virtuelles étendu.

9.3.3 Difficultés rencontrées

De manière général, les difficultés rencontrées sont les suivantes (classifiées par ordre décroissant de consommation de temps) :

1. Bien définir le cadre des cas tests.
2. Temps nécessaire pour les réalisations et spécifications.

9.4 Conclusion

De manière générale, les objectifs de ce travail ont pu être atteints. La matière technique nécessaire au laboratoire de virtualisation de l'HEPIA pour commencer sa migration vers KVM a pu être apportée. Le nouvel environnement de virtualisation du laboratoire a été spécifié et sa mise en place par Cédric Penas a déjà débuté. L'évolution des infrastructures KVM du laboratoire vers des environnements complètement automatisés a aussi pu être discuté.

Cependant, un certain nombre d'objectifs annexes n'ont pu être réalisés dans le temps disponible. L'étude de sheepdog entreprise a montré un produit extrêmement novateur avec un énorme potentiel, mais pas encore assez stable. Les sorties très fréquentes de nouvelles versions rendaient toutes études ou analyses trop complexes à la vue des changements apportés à chaque version. Cependant, les lecteurs intéressés trouveront sur le site officiel [4] et sur les slides des conventions KVM [5] de très nombreuses informations.

Bien qu'un dépassement d'heures soit clairement visible, il a été possible de remplir les demandes des mandants de ce projet en fournissant une étude transversale d'un environnement KVM couvrant le moteur de virtualisation, les drivers virtio et la librairie libvirt indispensable à tout environnement KVM.

De plus, la première partie de ce document traite de la virutalisation et des solutions de virtualisations de manière large et historique afin de permettre au lecteur d'obtenir les clés nécessaires à l'étude de KVM et aller plus loin dans son analyse.

4. Site officiel de sheepdog : http://www.osrg.net/sheepdog/
5. Slides à propos de Sheepdog : http://www.linux-kvm.org/wiki/images/8/8d/2010-forum-sheepdog.pdf

Conclusion

A man who carries a cat by the tail learns something he can learn in no other way.
Mark Twain.

KVM, une technologie d'avenir

Ce projet a permis d'explorer de manière pointue KVM et a aidé à découvrir comment utiliser cette solution de virtualisation de manière étendue.

KVM est clairement une technologie d'avenir même si son administration avancée demande des connaissances du monde Linux. En effet, un administrateur avancé de serveurs de virtualisation KVM doit au minimum être familiarisé avec les domaines suivants :

- Administration de base d'une machine Linux (Configurations réseaux, SSH, etc.).
- Ordonnancement des tâches sous Linux et utilisation de l'ordonnanceur.
- Comprendre de manière générale comment Linux alloue et libère sa mémoire.
- Être capable d'utiliser les outils réseaux sous Linux (Tap, bridge, routage, firewall, etc.).
- Comprendre et être capable d'utiliser le système de gestion de volumes logiques LVM.

Les livres donnés en introduction couvrent tous et bien plus que ce qui est demandé dans ces critères. Les connaissances pour répondre aux critères montrés ci-dessus sont atteignables à l'aide des quatre tomes de Administration Linux présentées en introduction de ce document.

Ces recommandations concernent bien évidemment les administrateurs avancés. KVM est aussi utilisable avec des outils disponibles de base sur les distributions Linux actuelles. Ces outils permettent de créer un serveur ou une station de travail de virtualisation simple de manière très aisée avec autant de simplicité que ce que propose des concurrents commerciaux de KVM. Cette flexibilité est l'une des forces de KVM avec un seul produit pour une grande variété d'utilisations possibles. KVM étant uniquement un moteur de virtualisation, ce sont les outils administrant KVM qui décident à qui se destine l'environnement de virtualisation construit.

La force de KVM est comme l'a montré son analyse, son intégration directe dans le noyau de Linux. Cette intégration synchrone permet à KVM d'être la solution de virtualisation actuelle de type 1 (hyperviseur de type 1) avec le plus grand support matériel et la plus simple à mettre à jour. De plus, son intégration et ses

outils graphiques permettent à KVM d'être une solution en concurrence avec les produits de virtualisation de type 2 (hyperviseur de type 2) simple, efficace et à la portée de tous les utilisateurs.

L'analyse du code source de KVM a révélé une solution écrite de manière efficace comprenant très peu de code comparativement aux fonctionnalités (en développement, en général, moins de code signifie : maintenance du code, moins de failles et moins de bugs). Bien que concis, le code n'en est pas moins complexe par les principes mise en oeuvre et les choix d'implémentations audacieux effectués. L'intégration au noyau Linux, l'ajout d'un mode de fonctionnement à celui-ci ainsi que la réutilisation maximum des structures et outils que le noyau proposait sont des choix audacieux qui ont demandé beaucoup de travail, mais permettent à KVM d'être construit sur de solides fondations pour des développements futurs. Ces choix aideront KVM à devenir un féroce concurrent sur le marché de la virtualisation en pleine croissance.

La conclusion de l'analyse de KVM et des drivers virtio (inclus par défaut dans le noyau Linux) a permis de se rendre compte des améliorations qui devraient intervenir dans les mois et années à venir. Les performances devraient s'accroître autant avec l'amélioration du matériel (de plus en plus d'aide à la virtualisation dans de plus en plus de matériels différents) que dans les améliorations software possibles (Améliorations des drivers paravirtualisés, diminution des pertes dans les différents changements de mode du noyau, etc.).

Depuis le 17 mai 2011, `IBM`, `Intel`, `Red Hat`, `HP`, `BMC Software`, `Eucalyptus Systems`, et `SUSE` ont créé un groupement (Open Virtualization Alliance) [6] dans le but de développer et de promouvoir KVM comme la solution de virtualisation du futur. Cette coalition de grands noms permettra à cette technologie de reposer sur des références solides du marché informatique mondial. Rappelons que comme expliqué dans le chapitre consacré à l'historique des solutions de virtualisation, IBM a choisi KVM comme solution pour ces nouveaux mainframes.

Idées et continuité

Le monde entourant les solutions de virutalisation basées sur KVM est très large et permettrait encore beaucoup d'études. A la demande des mandants de ce projet, des idées de projets de diplômes, de semestre ou d'étude pour les assistants du laboratoire sont listées ci-dessous. Ces projets permettraient de créer une base d'expérience solide du monde KVM.

Les interfaces graphiques et solutions de gestion de parcs virtuels

- Étudier la solution d'administration graphique Archipel [7].
- Étudier la solution Promox(Permettant la création et l'administration de machines virtuelles [8].
- Étudier la solution Ganeti permettant la création de l'administration de parcs de machines virtuelles [9].
- Étudier la solution de gestion de cloud Eucalyptus [10].
- Développer ou améliorer une solution de management existantes en fonctions de nouveaux besoins.

Point de détail de KVM

- Analyser de manière détaillée les choix d'utilisations des instructions de virtualisation modernes et la gap entre ce qui existe et ce qui est utilisé dans KVM.
- Analyser et mettre à l'épreuve les drivers virtio pour en quantifier les limites et améliorations possibles de manière détaillée.

6. Site officiel du Open Virtualization Alliance : http://www.openvirtualizationalliance.org/
7. Site officiel de Archipel : http://archipelproject.org/
8. Site officiel de Promox : https://www.proxmox.com/
9. Site officiel de Ganeti : https://code.osuosl.org/projects/ganeti-webmgr
10. Site officiel de eucalyptus : http://www.eucalyptus.com/

Management d'environnements de virtualisations

- Construire les bonnes pratiques modernes de la virtualisation avec KVM.
- Analyser et comparer des solutions de stockages classiques et novatrices pour KVM.
- Créer une formation KVM à destination des entreprises pour en faciliter la démocratisation.

La virtualisation et divers

- Étude approfondie du fonctionnement des assistances matérielles à la virtualisation dans le but de comprendre les enjeux des implémentations les utilisant.
- Établissement d'un comparatif qualitatif et quantitatif à long terme des approches KVM et VMWare.

Remerciements

- Le professeur Gerald Litzistorf pour son encadrement et ses conseils sur la direction à prendre avec ce travail.
- Yann Souchon pour son encadrement et ses conseils.
- Cédric Penas pour les réalisations pratiques dans le cadre de ce travail au laboratoire de virtualisation de l'HEPIA.
- Lionel Schaub actuel diplômant bachelor au laboratoire de virtualisation de l'HEPIA pour les résultats intermédiaires de son travail de diplôme qui permettaient de valider des hypothèses d'analyses.
- Richard Jones (Emerging Technologies architect) chez Red Hat pour les réponses précises à des problèmes rencontrés avec les outils de virtualisation KVM.
- Adam Litke (Virtualisation architect et créateur de MOM) chez IBM pour son assistance avec MOM et la problématique de l'overcommitement.
- Jonathan Mercier (Développeur et contributeur pour Fedora) pour son aide dans l'avancement de l'intégration de MOM dans Fedora.
- Anouk Kundig pour son aide à la relecture de parties du document.
- Michael Gingins pour son aide à la relecture de parties du document.
- Toutes les personnes disponibles sur les canaux IRC #fedora, #fedora-fr, #fedora-devel, #fedora-devel-fr, #kvm, #qemu pour leurs réponses à de multiples questions sur l'implémentation et l'architecture de KVM.

Conclusion personnel de projets

Ce projet a permis d'étudier une technologie a priori novatrice et à la mode pour finalement se rendre compte que fondamentalement, elle n'était pas si moderne (Voir chapitre La virtualisation, une problématique globale et historique) que ne le laisserait entendre les entreprises commerciales éditrices de solutions de virtualisation.

Il est intéressant de pouvoir étudier une technologie quand elle est encore jeune et en pleine définition. Actuellement, KVM est plébiscité par des grands noms de l'informatique (`RedHat`, `IBM`, etc.) mais, reste encore peu développée dans le milieu professionnel (Le cas RedHat mis à part) et libre. Par exemple, jusqu'à il y a peu, sauf Fedora (qui le fait depuis maintenant plusieurs années) et `Red Hat`, KVM n'était pour beaucoup pas encore la solution de virtualisation par défaut proposée aux utilisateurs de Linux.

Aujourd'hui, après avoir analysé le code source de KVM et les principaux composants annexe nécessaires que sont libvirt et virtio, il est possible de dire (malgré les améliorations possibles expliquées dans le chapitre

sur l'architecture de KVM) que KVM est une solution prête pour la production. Cependant, son administration et son utilisation avancée demande encore beaucoup de travail en ligne de commandes et de connaissance du monde linux. Cette problématique devrait se réduire avec le temps, l'adoption de plus en plus large de la solution devant créer une demande pour des outils d'administrations évolués.

L'environnement dans lequel ce travail a été réalisé fut lui aussi particulièrement intéressant parce que très libre. Le travail d'explorations de nouvelles technologies a pour avantage qu'il est libéré de certaines contraintes et permet d'embrasser complètement un domaine dans un but d'acquisition de savoir, de connaissance et de références plus qu'autre chose.

Résumé personnel du parcours de l'auteur

M. Gérald Litzistorf professeur responsable de l'encadrement ce travail était intéressé à ce que je fasse une description de mon parcours personnel (Académique, professionnel, autonome) lié au monde de Linux et de l'informatique. Ce court chapitre présente donc mon parcours personnel.

Parcours académique

J'ai commencé mon CFC (4 ans sans maturité) d'informaticien au CPLN (Centre professionnel du littoral Neuchâtelois) en 2000 en sortant de l'école (Section pré-professionnel, VSO dans le canton de Vaud). J'ai terminé mon CFC en 2004 avec un travail de diplôme consistant à développer une sonde de régulation basée sur un processeur Atmel (programmation en C). Cette sonde devait provoquer l'arrêt d'une machine en cas dépassement d'une hystérèse haute et le redémarrage quand une hystérèse basse était atteinte.

A la suite de mon CFC, j'ai commencé une formation de Technicien ES en informatique et télécommunication (CPLN). Très intéressé au monde des systèmes, j'ai néanmoins principalement travaillé sur des projets d'informatique embarquée dont deux principaux :

- Un périphérique équipé d'un GPS et d'un processeur PIC (programmation assembleur) devant enregistrer sur une carte mémoire les coordonnées et informations d'une randonnée. Ces informations récupérées, cette carte permettait de dessiner (logiciel sur PC) le parcours effectué sur une carte avec des informations sur les vitesses moyennes.
- Un système de domotique équipé de deux capteurs de températures. Ce système devait être interrogeable avec un navigateur web (Il a fallu créer le serveur web, le traitement des sondes, le système de base, etc.) sur un processeur PIC (programmation en C).

Commencant à être intéressé par la sécurité, j'ai réalisé pour mon travail de diplôme un système gérant l'intégrité des informations utilisateurs dans un Active Directory.

En 2006, j'ai commencé une formation d'ingénieur à la HEIG-VD avec une forte passion pour la sécurité dont j'ai teinté la plupart de mes projets. Mon travail de bachelor a été effectué pour mon employeur actuel (NetGuardians). Il s'agissait de développer un système de gestion et de visualisation des événements de sécurité survenant dans un système de messagerie.

Actuellement, je termine mon Master où j'ai choisi un maximum de cours dans le domaine de la sécurité. Une fois ce projet de semestre terminé, il ne me restera plus que ma thèse que je réaliserai dans le domaine des métriques de sécurité chez NetGuardians.

Parcours professionnel

Ma première expérience professionnelle était celle de livreur du journal l'express (équivalent du 24h pour les Neuchâtelois) le matin. Les horaires étant réellement peu adaptés aux études (03h-07h), j'ai commencé un travail à la médiathèque du CPLN en même temps que mes études de technicien.

J'ai aussi effectué beaucoup de petits travaux dans le domaine informatique (comme la plupart des étudiants en informatique) composés en grande partie d'installations et de réparations d'ordinateurs et de petit réseaux.

J'ai aussi travaillé pour l'entreprise DKProb afin de les aider à gérer leurs problèmes liés à l'informatique (Sécurité, systèmes de sauvegardes, procédures liées à l'IT, etc.).

Actuellement, je travaille en tant que Security Egineer pour l'entreprise NetGuardians, une entreprise active dans le log-management et les domaines liés (Compliance, suivi des systèmes, retours sur incidents, etc).

Mon parcours Linux.

J'ai découvert Linux avec Debian pendant ma seconde années de CFC (2001). Distribution que j'apprécie toujours et que j'ai utilisé jusqu'en 2005 où j'ai découvert le monde RedHat et Fedora vers lequel je me suis tourné et auquel je suis resté fidèle. J'ai commencé à m'intéresser beaucoup plus à Linux en 2006 où j'ai commencé à contribuer sur des projets open-source (petites contributions et patchs).

Aujourd'hui, j'essaie de contribuer autant qu'il m'est possible au projet Fedora et à divers autres projets open-source.

Dernièrement, j'ai participé à la réalisation du magazine gratuit Muffin N°3 [11] traitant du monde Linux et de l'open-source. J'ai aussi participé à la création du groupe Pycured [12] qui devrait voir débuter ces activités dans le monde de la sécurité et de l'open-source dans le courant de cet été.

11. Site officiel du magazine Muffin : http://mag.fedora-fr.org/wiki/Accueil
12. Site officiel du groupe Pycured : https://pycured.com

Bibliographie

[1] Christophe Blaess. *Développement système sous Linux*. Eyrolles, 2010.

[2] Jean-François Bouchaudy. *Linux Administration - Tome 1 - Les bases de l'administration système*. Eyrolles, 2009.

[3] Jean-François Bouchaudy. *Linux Administration - Tome 2 - Administration système avancée*. Eyrolles, 2009.

[4] Jean-François Bouchaudy. *Linux Administration - Tome 3 - Sécuriser un serveur Linux*. Eyrolles, 2010.

[5] Jean-François Bouchaudy. *Linux Administration - Tome 4 - Les services applicatifs Internet: Web, email, FTP*. Eyrolles, 2010.

[6] BTM.geek. « KVM Virtio network performance ». http://blog.loftninjas.org/2008/10/22/kvm-virtio-network-performance/.

[7] IBM Da Shuang He. « Create a KVM-based virtual server ». http://www.ibm.com/developerworks/linux/library/l-kvm-virtual-server/index.html.

[8] IBM M. Tim Jones. « Anatomy of a Linux hypervisor ». http://www.ibm.com/developerworks/linux/library/l-hypervisor/index.html.

[9] IBM M. Tim Jones. « Anatomy of the libvirt library ». http://www.ibm.com/developerworks/linux/library/l-libvirt/index.html.

[10] IBM M. Tim Jones. « Discover the Linux Kernel Virtual Machine ». http://www.ibm.com/developerworks/linux/library/l-linux-kvm/index.html.

[11] IBM M. Tim Jones. « Linux virtualization and PCI passthrough ». http://www.ibm.com/developerworks/linux/library/l-pci-passthrough/index.html.

[12] IBM M. Tim Jones. « Virtio: An I/O virtualization framework for Linux ». http://www.ibm.com/developerworks/linux/library/l-virtio/index.html.

[13] IBM M. Tim Jones. « Virtual Linux ». http://www.ibm.com/developerworks/library/l-linuxvirt/index.html.

[14] IBM M. Tim Jones. « Virtualization with coLinux ». http://www.ibm.com/developerworks/linux/library/l-virtualization-colinux/index.html.

[15] IBN M. Tim Jones. « Anatomy of the libvirt library ». http://www.ibm.com/developerworks/linux/library/l-virtual-networking/index.html.

[16] IBN M. Tim Jones. « Inside the Linux scheduler ». http://www.ibm.com/developerworks/linux/library/l-scheduler/.

[17] Richard Jones. « Virtio balloon ». https://rwmj.wordpress.com/2010/07/17/virtio-balloon/.

[18] Michael Kerrisk. *The Linux Programming Interface*. no starch press, 2010.

[19] IBM Adam Litke. « Manage resources on overcommitted KVM hosts ». http://www.ibm.com/developerworks/linux/library/l-overcommit-kvm-resources/index.html.

[20] Robert Love. *Linux Kernel Development*. Addison Wesley, 2010.

[21] lwn.net. « KVM virtio balloon driver ». https://lwn.net/Articles/265207/.

[22] Frank Mayer, Karl Macmillan, and David Caplan. *SELinux by example*. Prentice Hall, 2010.

[23] OSDEV. « WIKI OS DEV ». http://wiki.osdev.org/Main_Page.

[24] qemu buch. « QEMU and Kernel-based Virtual Machine ». http://qemu-buch.de/e/Content.

[25] Michael Rash. *Linux Firewall, attack detection and response with IPTABLES*. no starch press, 2007.

[26] Mulyadi Santosa. « An enlightenment about shadow page table ». http://the-hydra.blogspot.com/2006/12/enlightenment-about-shadow-page-table.html.

[27] Amit Singh. « An Introduction to Virtualization ». http://www.kernelthread.com/publications/virtualization/.

[28] Wikipedia. « Timeline of virtualization ». http://en.wikipedia.org/wiki/Timeline_of_virtualization_development.

[29] WMware. « Transparent Paravirtualization (VMI) ». http://www.vmware.com/technical-resources/interfaces/paravirtualization.html.

[30] www.linux kvm.com. « How do you use e1000 option on a windows Guest ». http://www.linux-kvm.com/content/how-do-you-use-e1000-option-windows-guest.

[31] www.semanticlab.net. « KVM increase CPU and RAM during runtime ». http://www.semanticlab.net/index.php/KVM_increase_CPU_and_RAM_during_runtime.

Table des matières