

首本Linux KVM虚拟化技术专著，由Intel虚拟化技术部门资深虚拟化技术专家和KVM研究者撰写

系统介绍KVM虚拟机的功能、特性和使用方法，深入剖析KVM虚拟机的核心技术和原理



任永杰 单海涛 著

KVM: Principles and Practices

KVM虚拟化技术 实战与原理解析



机械工业出版社
China Machine Press

KVM虚拟化技术： 实战与原理解析

任永杰 单海涛 著



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

KVM虚拟化技术: 实战与原理解析 / 任永杰, 单海涛著. —北京: 机械工业出版社, 2013.10

ISBN 978-7-111-43900-4

I. K… II. ①任… ②单… III. 虚拟处理机 IV. TP338

中国版本图书馆CIP数据核字 (2013) 第209229号

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问 北京市展达律师事务所

首本 Linux KVM 虚拟化技术专著, 由 Intel 虚拟化技术部门资深虚拟化技术专家和国内 KVM 技术的先驱者撰写, 权威性毋庸置疑。在具体内容上, 本书不仅系统介绍了 KVM 虚拟机的功能、特性和使用方法, 而且还深入地剖析了 KVM 虚拟机的核心技术和工作原理, 对 KVM 做了全面而透彻的讲解。

全书一共 9 章: 第 1 章介绍了云计算和虚拟化的概念, 并对 KVM 等几种流行的虚拟化技术做了比较; 第 2 章介绍了 KVM 的基本架构、QEMU 的作用以及 Intel 的硬件虚拟化技术; 第 3 章详细讲解了 KVM 依赖的硬件环境配置, 编译和安装 KVM 和 QEMU 的步骤与方法, 以及 KVM 客户机的启动; 第 4 章深入讲解了 KVM 的基础功能, 包括虚拟的 CPU、内存、存储、网络、图形显示等的配置和基本原理, 以及 CPU 和内存的过载使用; 第 5 章讲解了 KVM 的高级功能, 包括半虚拟化驱动 virtio、设备直接分配 VT-d、热插拔、动态迁移、嵌套虚拟化、KSM、透明大页、KVM 安全技术、QEMU 监控器、QEMU 命令行参数以及从物理机或其他虚拟机迁移到 KVM 的方法; 第 6 章介绍了管理 KVM 虚拟化的上层软件; 第 7 章介绍了 RHEL 等所有流行的 Linux 发行版中的 KVM 虚拟化功能的使用; 第 8 章首先介绍了虚拟化性能测试, 然后详细介绍了对 KVM 虚拟化的 CPU、内存、网络、磁盘 I/O 等重要组件进行性能测试的方法、工具和步骤; 第 9 章介绍了 Linux、KVM、QEMU 等开源社区的情况, KVM、QEMU 和 KVM 单元测试代码的基本结构, 以及如何向 QEMU/KVM 开源社区贡献自己的代码和如何提交 KVM 相关的 bug。

HZ BOOKS
华章图书

机械工业出版社 (北京市西城区百万庄大街22号 邮政编码 100037)

责任编辑: 姜 影

印刷

2013年10月第1版第1次印刷

186mm×240mm·26.5印张

标准书号: ISBN 978-7-111-43900-4

定 价: 79.00元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzsj@hzbook.com

前 言

为什么要写这本书

近几年，云计算大潮风起云涌，Amazon 的 AWS 是公共云计算平台的先驱和领导者，Google 的 GAE、GCE 也取得了一定的成绩，国内大型互联网公司也分别推出自己的云计算服务，如阿里云、腾讯云、盛大云、新浪 SAE、百度 BAE 等。云计算已成为 IT 技术领域最热门的词汇之一，现在很多互联网服务都声称自己是“云”服务。在云计算发展的大背景下，支撑着云计算服务的最底层、最基础的虚拟化技术也得到了快速发展，成为技术研究和应用的热点之一。

在虚拟化领域也有很多的技术和产品种类，包括传统的老牌虚拟化软件 VMware、Microsoft 的 Hyper-V、Citrix 的 XenServer/XenClient、Oracle 的 VirtualBox 等，还有在 Linux 平台上比较流行的开源虚拟化技术 KVM、Xen 等，甚至包括 Linux 上的轻量级虚拟化技术 LXC。随着 Linux 服务器广泛地应用在互联网等热门行业，KVM 作为 Linux 内核原生的虚拟化技术受到了开发者和用户广泛的关注，并且已经得到了较大规模的应用，如 Google 的公有云计算引擎 GCE 就是在底层完全使用 KVM 作为虚拟化技术来实现的。

KVM 虚拟机最初是由以色列一家创业公司 Qumranet 员工 Avi Kivity 等人开发，于 2006 年 8 月完全开源并推向 Linux 内核社区，当年 10 月被 Linux 社区接受。2007 年 2 月发布的 Linux 2.6.20 是第一个带有 KVM 模块的 Linux 内核正式发布版本。Red Hat 公司于 2008 年 9 月正式将 Qumranet 公司收购，接着投入了较多的资源到 KVM 虚拟化的开发中。在 2010 年 11 月，Red Hat 公司发布的 RHEL 6 中完全用 KVM 替换了 RHEL 5 中默认支持的 Xen，让 KVM 成为 RHEL 操作系统默认的虚拟化方案。

KVM 必须依赖 CPU 提供的硬件虚拟化技术，以 Intel、AMD 为代表的 x86 硬件平台在最近几年也推出了很多与虚拟化相关的硬件特性，包括最初基本的 CPU 的 VT 支持和 EPT、VPID，以及 I/O 设备的 VT-d、SR-IOV，还包括最新的 APIC-v、Shadow VMCS 等特性。随着 x86 硬件对虚拟化技术的支持越来越成熟，KVM 在最新的硬件平台上的虚拟化效率也得到很大的提高。

除了硬件方面的虚拟化技术逐渐成熟之外，KVM 作为 Linux 内核虚拟机，受到许许多多 Linux 软件开发者的青睐。Red Hat 公司无疑是 KVM 开发中最大的一股力量，很多 Red Hat

工程师在 KVM、QEMU、libvirt 等开源社区中成为核心开发成员。除了 Red Hat 之外，还有许多知名公司都有不少软件开发者为 KVM 贡献自己的代码，如 IBM、Intel、Novell、AMD、Siemens、华为等。除了这些大公司的开发者之外，还有许多小公司和个人独立开发者活跃在 KVM 开源社区，为 KVM 开发代码或者做测试工作。

在硬件方面的虚拟化支持和软件方面的功能开发、性能优化的共同作用下，目前 KVM 虚拟化技术已经拥有非常丰富的功能和非常优秀的性能。而且，KVM 虚拟化的上层应用软件和云计算平台（如 libvirt、Ovirt、Virt-Manager、OpenStack 等）也是日渐成熟。因此，KVM 已经成为个人和企业在进行系统虚拟化或云平台建设的底层虚拟化时一个非常不错的选择。

在 2012 年，我们接受了机械工业出版社杨福川老师的邀请，决定写一本书来专门介绍 KVM 虚拟化的功能和原理，经过 1 年多的精心准备和努力写作，本书终于顺利出版。

本书所有作者在虚拟化技术领域有多年的学习和工作经验，积累了比较丰富的知识和经验。任永杰在 2011 年年初加入 Intel 的云计算与虚拟化部门，专注于 Linux 开源虚拟化的功能测试、性能分析以及与开源社区交流等工作。单海涛于 2007 年加入 Intel 虚拟化部门，一直从事 KVM、Xen 等虚拟化软件的开发及开发团队的管理。本书内容由任永杰统筹规划，单海涛编写了本书的第 1、2 两章，其余的第 3 至 9 章由任永杰编写。

在国内，本书是目前唯一一本专门介绍 KVM 虚拟化技术的书籍，而且在英文书籍中也没有任何一本书是在专门介绍 KVM 技术。本书可以让大家对 Linux 内核虚拟机——KVM 的原理、技术、功能和性能有较为详细的了解，也希望能通过本书让更多的读者在学习和工作中实际应用 KVM 虚拟化技术，并且能够参与到 Linux、KVM、QEMU 等开源社区中去。

读者对象

本书适合对 Linux 下虚拟化或云计算基础技术感兴趣的读者，包括 Linux 运维工程师、KVM 开发者、云平台开发者、虚拟化方案决策者、KVM 的用户以及其他对 KVM 虚拟机感兴趣的计算机爱好者。希望本书对这些读者了解 KVM 提供以下帮助。

- ❑ 运维工程师：了解 KVM 的使用方法、功能和基本的性能数据，能够搭建高性能的 KVM 虚拟化系统并将其应用于生产环境中。
- ❑ KVM 开发者：了解 KVM 的基本原理和功能，了解其基本用法和一些调试方法，以及如何参与到 KVM 开源社区中去贡献代码。
- ❑ 云平台开发者：了解底层 KVM 虚拟化的基本原理和用法，以促进云平台上层应用的开发和调试的效率。
- ❑ 虚拟化方案决策者：了解 KVM 的硬件环境需求和它的功能、性能概况，以便在虚拟化技术选型时做出最优化的决策。
- ❑ 普通用户：了解 KVM 的功能和如何使用 KVM，用 KVM 虚拟化技术来促进相应的学习、开发和测试。

如何阅读本书

本书分为 9 章，具体介绍如下内容：

第 1 章简单介绍了云计算的概念和技术，然后简单介绍了目前比较流行的包括 KVM 在内的几种虚拟化技术。

第 2 章简单介绍了 Linux 的基本历史、KVM 的基本架构、QEMU 的作用以及 Intel 的硬件虚拟化技术概况。

第 3 章详细介绍了 KVM 依赖的硬件环境配置、编译以及安装 KVM 和 QEMU 的步骤与方法，最后演示了如何启动一个 KVM 客户机。

第 4 章详细介绍了客户机系统的 CPU、内存、存储、网络、图形显示等方面的配置和基本原理，以及 CPU 和内存的过载使用。

第 5 章详细介绍了除了第 4 章的基础功能之外的高级功能，包括半虚拟化驱动 virtio、设备直接分配 VT-d、热插拔、动态迁移、嵌套虚拟化、KSM、透明大页、KVM 安全技术、QEMU 监控器、QEMU 命令行参数以及从物理机或其他虚拟机迁移到 KVM 的方法。在介绍每个功能时，都尽量对其背后的原理和技术做了简单的介绍。

第 6 章介绍了管理 KVM 虚拟化的上层软件，包括 libvirt、virt-manager、virt-viewer、virt-install 以及 OpenStack 云计算平台。

第 7 章分别介绍了 RHEL、Fedora、SLES、OpenSuse、Ubuntu 等 Linux 发行版中的 KVM 虚拟化功能的使用。

第 8 章首先简单介绍了虚拟化性能测试，然后详细介绍对 KVM 虚拟化的 CPU、内存、网络、磁盘 I/O 等重要组件进行性能测试的方法、常见工具和具体测试步骤，并且分享了作者在实际性能测试实验中得到的一些数据供读者参考，以便读者对 KVM 的性能有基本的认识。

第 9 章简单介绍了 Linux、KVM、QEMU 等开源社区的基本情况，也简单介绍了 KVM、QEMU 和 KVM 单元测试代码的基本结构，接着详细介绍了如何向 QEMU/KVM 开源社区贡献自己的代码以及如何提交 KVM 相关的 bug。

本书中每章后面都有“本章小结”，部分章节后面有“本章注释和参考阅读”。“本章注释”是本章正文中一些需要详细说明的地方，“参考阅读”中包含了笔者推荐的与本章内容相关的一些参考资料供有兴趣的读者对相关内容做更多的了解。

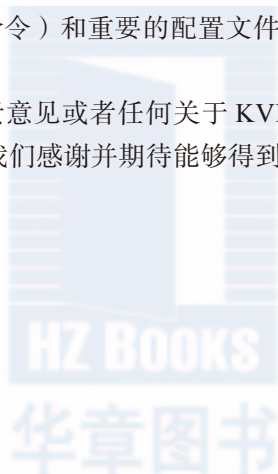
读者可以根据自己的兴趣和已经掌握的知识来有选择地阅读各个章节。如果对 KVM 基本没有什么了解，建议按本书顺序阅读，通读一遍之后再对感兴趣的章节进行仔细阅读。今后可能会经常使用 KVM 的读者，在阅读本书之时，可以根据书中示例进行实际操作，如果是开发者也可以查看相应的源代码。另外，如果对某些章节中的内容比较感兴趣，建议阅读每章末尾笔者推荐的一些参考资料。

勘误和支持

KVM、QEMU 等开源社区非常活跃，QEMU/KVM 发展迅速，经常有新的功能加进去或者原有功能被改进，特别是 QEMU 命令行参数很可能会有所改动。因此本书中 QEMU 命令行参数只能适用于本书中提及的 QEMU 版本，若使用不同的版本，命令行参数可能并不完全相同。另外，本书中的 QEMU 命令行示例一般都没有添加“-enable-kvm”参数，这是由于笔者写作本书时使用的 qemu-kvm 是默认启用 KVM 加速的，如果读者使用普通的 QEMU 来启动 KVM 客户机，就请根据需要自行在 QEMU 命令行中添加“-enable-kvm”参数。

由于 KVM 和 QEMU 的发展变化比较快，加之作者的技术水平有限和编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者朋友批评指正。读者朋友对本书相关内容有任何疑问、批评和建议，都可以通过作者之一（任永杰）的博客网站 <http://smilejay.com/> 进行讨论，作者会尽力回复每一条留言信息并帮助读者得到满意的答案。全书中涉及的示例代码程序（不包含单行的命令）和重要的配置文件，都可以从网站 <https://github.com/smilejay/kvm-book> 查看和下载。

如果读者朋友们有更多的宝贵意见或者任何关于 KVM 虚拟化技术的讨论，欢迎发送电子邮件至 smile665@gmail.com，我们感谢并期待能够得到朋友们的真挚反馈。



致 谢

首先要感谢 Linus Torvalds 这样的“大神”级别的 Linux 程序员，是他们创造了一个伟大的操作系统内核——Linux。

其次要感谢 Avi Kivity 等创建 KVM 项目的程序员，如果没有 KVM 项目，自然就不会有本书的存在。

感谢 Intel 开源技术中心的云计算与虚拟化部门，我们在这里学到了很多与 Linux 和虚拟化相关的技术。特别感谢该部门的张献涛，他是 KVM IA64 架构的维护者，在本书写作过程中，给了我们很多帮助和建议。

非常感谢机械工业出版社华章公司的编辑杨福川和姜影老师，在这一年多的时间中始终支持本书的写作，在他们的鼓励和帮助下，我们才能顺利地完成本书的写作。

最后感谢我们的爸爸、妈妈，感谢你们将我们养育成人并给予最无私的爱！

谨以此书献给我们的家人，以及众多喜欢 Linux、KVM 等开源软件的朋友们！

任永杰 单海涛

HZ BOOKS
华章图书

目 录

前言

第 1 章 虚拟化与云计算	1
1.1 云计算概念	1
1.1.1 发展历史和现状	1
1.1.2 概念	2
1.1.3 云计算模式	3
1.2 云计算技术	4
1.2.1 Map/Reduce	4
1.2.2 资源管理平台	5
1.2.3 虚拟化	5
1.3 虚拟化技术	5
1.3.1 软件虚拟化和硬件虚拟化	6
1.3.2 准虚拟化与全虚拟化	8
1.4 KVM简介	9
1.4.1 KVM的历史	9
1.4.2 KVM功能概览	9
1.4.3 KVM的前景	11
1.5 Xen简介	11
1.5.1 Xen的历史	11
1.5.2 Xen功能概览	12
1.5.3 Xen的前景	13
1.6 其他虚拟化方案简介	13
1.6.1 VMware	14
1.6.2 VirtualBox	14
1.6.3 Hyper-V	15

1.7	本章小结	15
1.8	本章注释	15
第 2 章	KVM 原理简介	17
2.1	Linux操作系统简介	17
2.2	虚拟化模型	18
2.3	KVM架构	19
2.4	KVM模块	21
2.5	QEMU设备模型	22
2.6	Intel虚拟化技术	22
2.7	本章小结	23
第 3 章	构建 KVM 环境	24
3.1	硬件系统的配置	24
3.2	安装宿主机Linux系统	26
3.3	编译和安装KVM	28
3.3.1	下载KVM源代码	28
3.3.2	配置KVM	30
3.3.3	编译KVM	33
3.3.4	安装KVM	35
3.4	编译和安装qemu-kvm	37
3.4.1	下载qemu-kvm源代码	37
3.4.2	配置和编译qemu-kvm	38
3.4.3	安装qemu-kvm	39
3.5	安装客户机	40
3.6	启动第一个KVM客户机	42
3.7	本章小结	43
3.8	本章注释和参考阅读	44
第 4 章	KVM 核心基础功能	45
4.1	硬件平台和软件版本说明	45
4.2	CPU配置	48

4.2.1	vCPU的概念	48
4.2.2	SMP的支持	49
4.2.3	CPU过载使用	54
4.2.4	CPU模型	55
4.2.5	进程的处理亲和性和vCPU的绑定	57
4.3	内存配置	61
4.3.1	内存设置基本参数	61
4.3.2	EPT和VPID简介	64
4.3.3	大页 (Huge Page)	66
4.3.4	内存过载使用	68
4.4	存储配置	70
4.4.1	存储配置和启动顺序	70
4.4.2	qemu-img命令	75
4.4.3	QEMU支持的镜像文件格式	78
4.4.4	客户机存储方式	80
4.5	网络配置	82
4.5.1	QEMU支持的网络模式	83
4.5.2	使用网桥模式	85
4.5.3	使用NAT模式	90
4.5.4	QEMU内部的用户模式网络	98
4.5.5	其他网络选项	102
4.6	图形显示	103
4.6.1	SDL的使用	103
4.6.2	VNC的使用	105
4.6.3	VNC显示中的鼠标偏移	112
4.6.4	非图形模式	113
4.6.5	显示相关的其他选项	115
4.7	本章小结	116
4.8	本章注释和参考阅读	116
第 5 章	KVM 高级功能详解	120
5.1	半虚拟化驱动	120

5.1.1	virtio概述	120
5.1.2	安装virtio驱动	122
5.1.3	使用virtio_balloon	128
5.1.4	使用virtio_net	133
5.1.5	使用virtio_blk	137
5.1.6	kvm_clock配置	138
5.2	设备直接分配 (VT-d)	140
5.2.1	VT-d概述	140
5.2.2	VT-d环境配置	141
5.2.3	VT-d操作示例	150
5.2.4	SR-IOV技术	160
5.3	热插拔	170
5.3.1	PCI设备热插拔	170
5.3.2	PCI设备热插拔示例	171
5.3.3	CPU和内存的热插拔	176
5.4	动态迁移	177
5.4.1	动态迁移的概念	177
5.4.2	动态迁移的效率和应用场景	178
5.4.3	KVM动态迁移原理和实践	180
5.4.4	VT-d/SR-IOV的动态迁移	187
5.5	嵌套虚拟化	188
5.5.1	嵌套虚拟化的基本概念	188
5.5.2	KVM嵌套KVM	188
5.6	KSM技术	191
5.6.1	KSM基本原理	191
5.6.2	KSM操作实践	193
5.7	KVM其他特性简介	197
5.7.1	1GB大页	197
5.7.2	透明大页	200
5.7.3	AVX和XSAVE	203
5.7.4	AES新指令	205
5.7.5	完全暴露宿主机CPU特性	210
5.8	KVM安全	212

5.8.1	SMEP	212
5.8.2	控制客户机的资源使用——cgroups	213
5.8.3	SELinux和sVirt	220
5.8.4	可信任启动——Tboot	228
5.8.5	其他安全策略	237
5.9	QEMU监控器	241
5.9.1	QEMU monitor的切换和配置	241
5.9.2	常用命令介绍	242
5.10	qemu-kvm命令行参数	249
5.10.1	回顾已用过的参数	249
5.10.2	其他常用参数	253
5.11	迁移到KVM虚拟化环境	261
5.11.1	virt-v2v工具介绍	261
5.11.2	从Xen迁移到KVM	262
5.11.3	从VMware迁移到KVM	265
5.11.4	从VirtualBox迁移到KVM	266
5.11.5	从物理机迁移到KVM虚拟化环境（P2V）	266
5.12	本章小结	267
5.13	注释和参考阅读	268
第 6 章	KVM 管理工具	272
6.1	libvirt	272
6.1.1	libvirt简介	272
6.1.2	libvirt的编译、安装和配置	275
6.1.3	libvirt和libvirtd的配置	281
6.1.4	libvirt域的XML配置文件	285
6.1.5	libvirt API简介	297
6.1.6	建立到Hypervisor的连接	299
6.1.7	libvirt API使用示例	303
6.2	virsh	308
6.2.1	virsh简介	308
6.2.2	virsh常用命令	310

6.3	virt-manager	316
6.3.1	virt-manager简介	316
6.3.2	virt-manager编译和安装	317
6.3.3	virt-manager使用	317
6.4	virt-viewer、virt-install和virt-top	324
6.4.1	virt-viewer	324
6.4.2	virt-install	325
6.4.3	virt-top	327
6.5	OpenStack云计算平台	327
6.5.1	OpenStack简介	327
6.5.2	使用DevStack安装和配置OpenStack开发环境	330
6.5.3	在OpenStack中使用KVM	332
6.6	本章小结	334
6.7	本章注释和参考阅读	334
第 7 章	Linux 发行版中的 KVM	336
7.1	RHEL和Fedora中的KVM	336
7.1.1	Red Hat、RHEL、Fedora和CentOS简介	336
7.1.2	RHEL中的KVM	337
7.1.3	Fedora中的KVM	338
7.2	SLES和openSUSE中的KVM	339
7.2.1	SLES中的KVM	339
7.2.2	openSUSE中的KVM	341
7.3	Ubuntu中的KVM	341
7.4	本章小结	343
7.5	本章注释和参考阅读	344
第 8 章	KVM 性能测试及参考数据	345
8.1	虚拟化性能测试简介	345
8.2	CPU性能测试	347
8.2.1	CPU性能测试工具	347
8.2.2	测试环境配置	349

8.2.3	性能测试方法	350
8.2.4	性能测试数据	352
8.3	内存性能测试	354
8.3.1	内存性能测试工具	355
8.3.2	测试环境配置	356
8.3.3	性能测试方法	356
8.3.4	性能测试数据	357
8.4	网络性能测试	360
8.4.1	网络性能测试工具	360
8.4.2	测试环境配置	361
8.4.3	性能测试方法	362
8.4.4	性能测试数据	364
8.5	磁盘I/O性能测试	367
8.5.1	磁盘I/O性能测试工具	367
8.5.2	测试环境配置	368
8.5.3	性能测试方法	369
8.5.4	性能测试数据	370
8.6	本章小结	373
8.7	本章注释和参考阅读	373
第9章	参与 KVM 开源社区	375
9.1	开源社区介绍	375
9.1.1	Linux开源社区	375
9.1.2	KVM开源社区	377
9.1.3	QEMU开源社区	377
9.1.4	其他开源社区	378
9.2	代码结构简介	379
9.2.1	KVM代码	379
9.2.2	QEMU代码	381
9.2.3	KVM单元测试代码	383
9.2.4	KVM Autotest代码	385
9.3	向开源社区贡献代码	386

9.3.1	开发者邮件列表	386
9.3.2	代码风格	387
9.3.3	生成patch	391
9.3.4	检查patch	394
9.3.5	提交patch	396
9.4	提交KVM相关的bug	399
9.4.1	通过邮件列表提交bug	399
9.4.2	使用bug管理系统提交bug	401
9.4.3	使用二分法定位bug	401
9.5	本章小结	404
9.6	本章注释和参考阅读	404



第1章 虚拟化与云计算

本书是一本以 KVM 为例，讲解虚拟化技术的书。可是看到这一章的标题，有读者会问：虚拟化为什么会跟云计算扯到一起去呢？其实，作为两个时下比较热门的技术，它们之间有着密不可分的千丝万缕的联系。本章将试图为读者理清它们之间的关系，从简单介绍云计算的概念开始，逐步解释虚拟化技术在云计算中的作用，并对当前流行的虚拟化解决方案进行一些简单的对比。

1.1 云计算概念

云计算是一个本世纪初才方兴未艾的热门名词，其本身是由 Google 前首席执行官埃里克·施密特（Eric Schmidt）在 2006 年 8 月 9 日的搜索引擎大会（SES San Jose 2006）上首次提出的。在此之后，云计算变得炙手可热，很多公司趁势宣传，号称自己是先进的云计算公司。加之业界也一直没有对云计算形成一个统一的定义，各种各样的云变得“乱花渐欲迷人眼”。抛开形形色色的宣传，本书试图为读者一步步理清云计算的真实面目。

1.1.1 发展历史和现状

虽然云计算的提出距离现在还只是短短的几年，但是云计算并不算是个彻底的新事物，因为早在它之前已经有各种类似或关联的概念被先后提出。早在 60 年代麦卡锡（John McCarthy）就提出了把计算能力作为一种像水和电一样的公用事业提供给用户。1999 年，Salesforce.com 提出的通过一个网站向企业提供企业级的应用的概念，已经颇有云计算的雏形。另外，20 世纪 90 年代，为了充分利用空闲的 CPU 资源，通过网络互联搭建平行分布式计算平台，诞生了网格计算，例如 1999 年出现的 SETI@home^[1] 科学研究项目。在网格计算中，将一个大型的计算任务拆分，分配给各个网格终端计算然后组合的思想，与云计算中的 Map/Reduce 技术不谋而合。

这些早期的概念虽然与云计算相关，但是要么碍于当时的计算机和网络技术，要么只是作为一种研究项目，始终没有能够被广泛推广并形成规模。然而，在本世纪初，计算机网络的大量普及和 Web 数据中心的建立，为云计算提供了必要的硬件准备。

在现有的被大家熟知并使用的共有云计算平台中，最著名的是由 Amazon（亚马逊）和 Google（谷歌）公司提供的服务。

Amazon 使用弹性计算云（EC2）和简单存储服务（S3）为企业提供计算和存储服务。

收费的服务项目包括存储服务器、带宽、CPU 资源以及月租费。月租费与电话月租费类似，存储服务器、带宽按容量收费，CPU 根据时长（小时）运算量收费。Amazon 把云计算做成一个大生意并没有花太长的时间：不到两年时间，在 Amazon 上注册的开发人员就达 44 万人，还有为数众多的企业级用户。云计算是 Amazon 增长最快的业务之一。Amazon 的财报数据显示，在其 2012 年收入中，云计算相关业务的收入已经高达 21 亿美元。

Google 则当数最大的云计算的使用者。Google 搜索引擎就建立在分布在全世界 200 多个地点、超过 100 万台服务器的支撑之上，这些设施的数量正在迅猛增长。Google 地球、地图、Gmail、Docs 等也同样使用了这些基础设施。对于 Google Docs^[2] 之类的应用，用户数据会保存在互联网上的某个位置，可以通过任何一个与互联网相连的系统十分便利地访问这些数据。Google 于 2008 年 4 月推出了 Google App Engine（GAE）云计算服务，GAE 是采用“平台即服务”（PaaS）的云计算模式，可以让第三方开发者在 GAE 平台上快速地开发自己的 Web 应用程序，而且 GAE 还将开发者的 Web 应用程序部署在 Google 全球的数据中心。Google 还于 2012 年 6 月底发布了自己的“基础设施即服务”（IaaS）模式的云计算平台——Google Compute Engine（GCE），GCE 的底层采用的是 KVM 虚拟化技术，它提供了与亚马逊 EC2 类似的云计算服务。

随着云计算的价值被不断发掘，越来越多的大公司开始加入到研究和部署云计算技术的阵营。

2008 年 2 月 1 日，IBM 宣布将在中国无锡的太湖新城科教产业园为中国的软件公司建立全球第一个云计算中心。

2008 年 7 月 29 日，雅虎、惠普和英特尔宣布一项涵盖美国、德国和新加坡的联合研究计划，推出云计算研究测试床，推进云计算。该计划要与合作伙伴创建 6 个数据中心作为研究试验平台，每个数据中心配置 1400 个至 4000 个处理器。这些合作伙伴包括新加坡资讯通信发展管理局、德国卡尔斯鲁厄大学 Steinbuch 计算中心、美国伊利诺伊大学香槟分校、英特尔研究院、惠普实验室和雅虎。

2010 年 7 月，美国国家航空航天局和包括 Rackspace、AMD、Intel、戴尔等支持厂商共同宣布“OpenStack”开放源代码计划，微软在 2010 年 10 月表示支持 OpenStack 与 Windows Server 2008 R2 的集成，而 Ubuntu 已把 OpenStack 添加到 Ubuntu 11.04 版本中。

2011 年 2 月，思科系统正式加入 OpenStack，重点研发 OpenStack 的网络服务。

1.1.2 概念

云计算自从提出，一直没有一个明确而统一的定义。维基百科对云计算做了如下的描述^[3]：云计算是一种通过因特网以服务的方式提供动态可伸缩的虚拟化的资源的计算模式（Cloud computing is a style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet）。美国国家标准与技术研究院（NIST）定

义：云计算是一种按使用量付费的模式，这种模式提供可用的、便捷的、按需的网络访问，进入可配置的计算资源共享池（资源包括网络、服务器、存储、应用和服务），这些资源能够被快速提供，只需投入很少的管理工作，或服务供应商进行很少的交互（Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction)。

也有人基于云端计算的实现方式，认为云计算是分布式计算技术的一种，其最基本的概念是，透过网络将庞大的计算处理程序自动分拆成无数个较小的子程序，再交给由多部服务器所组成的庞大系统经搜寻、计算分析之后将处理结果回传给用户。透过这项技术，网络服务提供者可以在数秒之内，处理数以千万计甚至亿计字节的信息，实现和“超级计算机”同样强大效能的网络服务。提供资源的网络被称为“云”。“云”中的资源在使用者看来是可以无限扩展的，并且可以随时获取，按需使用。

1.1.3 云计算模式

云计算是推动 IT 转向以业务为中心模式的一次重大变革。它着眼于运营效率、竞争力和快速响应等实际成果。这意味着 IT 的作用正在从提供 IT 服务逐步过渡到根据业务需求优化服务的交付和使用。这种全新的模式将以前的信息孤岛转化为灵活高效的资源池和具备自我管理能力的虚拟基础架构，从而以更低的成本和更好的服务的形式提供给用户。IT 即服务将提供业务所需要的一切，并在不丧失对系统的控制力的同时，保持系统的灵活性和敏捷性。

云计算的表现形式多种多样，简单的云计算在日常网络应用中随处可见，比如腾讯 QQ 空间提供的在线制作 Flash 图片，Google 的搜索服务，Google Docs 和 Google Apps 等。目前，云计算的主要服务形式有：SaaS（Software as a Service，软件即服务）、PaaS（Platform as a Service，平台即服务）和 IaaS（Infrastructure as a Service，基础设施即服务）。

1. 软件即服务

SaaS 提供商将应用软件统一部署在自己的服务器上，用户根据需求通过互联网向厂商订购应用软件服务，服务提供商根据客户所定软件的数量、时间的长短等因素收费，并且通过浏览器向客户提供软件的模式。这种服务模式的优势是，由服务提供商维护和管理软件，提供软件运行的硬件设施，用户只需拥有能够接入互联网的终端，即可随时随地使用软件。在这种模式下，客户不再像传统模式那样花费大量资金在硬件、软件、维护人员上，只需要支出一定的租赁服务费用，通过互联网就可以享受到相应的硬件、软件和维护服务，这是网络应用最具效益的营运模式。对于小型企业来说，SaaS 是采用先进技术的最好途径。

目前，Salesforce.com 是提供这类服务最著名的公司，Google Docs、Google Apps 和 Zoho Office 也属于这类服务。

2. 平台即服务

PaaS 把开发环境作为一种服务来提供。这是一种分布式平台服务，厂商提供开发环境、服务器平台、硬件资源等服务给客户，用户在其平台基础上定制开发自己的应用程序并通过其服务器和互联网传递给其他客户。PaaS 能够为企业或个人提供研发的中间件平台，提供应用程序开发、数据库、应用服务器、试验、托管及应用服务。

Google App Engine、Salesforce 的 force.com 平台、八百客的 800APP 是 PaaS 的代表产品。以 Google App Engine 为例，它是一个由 python 应用服务器群、BigTable 数据库及 GFS 组成的平台，为开发者提供一体化主机服务器及可自动升级的在线应用服务。用户编写应用程序并在 Google 的基础架构上运行就可以为互联网用户提供服务，Google 提供应用运行及维护所需要的平台资源。

3. 基础设施即服务

IaaS 即把厂商的由多台服务器组成的“云端”基础设施，作为计量服务提供给客户。它将内存、I/O 设备、存储和计算能力整合成一个虚拟的资源池为整个业界提供所需要的存储资源和虚拟化服务器等服务。这是一种托管型硬件方式，用户付费使用厂商的硬件设施。例如 Amazon Web 服务（AWS）、Google 的 Google Compute Engine、IBM 的 BlueCloud 等均将基础设施作为服务出租。

IaaS 的优点是用户只需低成本获得所需的硬件资源，按需租用相应计算能力、存储容量、网络带宽，而且省去了硬件运维方面的成本，大大降低了用户在硬件上的开销。

1.2 云计算技术

云计算的兴起，改变了现有的以本地计算为主的应用模型。用户不再需要付费购买软件并将其安装到本地的计算机中执行。取而代之，大量的计算任务，由客户端通过网络发起，在云计算提供商的数据中心的服务器集群上进行计算，其结果经由网络返回，在客户端进行呈现。新的计算模型的提出，必然伴随着新的问题需要解决。在云计算的环境下，不同厂商对如何有效地管理云端的资源，为用户提供快捷的计算进行了大量研究，提出和总结了一些行之有效的云计算技术。

1.2.1 Map/Reduce

Map/Reduce 是 Google 开发的编程模型，它是一种简化的分布式编程模型和高效的任务调度模型，用于大规模数据集（大于 1TB）的并行运算。严格的编程模型使云计算环境下的编程十分简单。MapReduce 模式的思想是将要执行的问题分解成 Map（映射）和 Reduce（化简）的方式，先通过 Map 程序将数据切割成不相关的区块，分配（调度）给大量计算机处理，达到分布式运算的效果，再通过 Reduce 程序将结果汇总输出。

Map/Reduce 编程模型适用于很多应用，例如，分布式搜索、分布式排序、机器学习、

基于统计的机器翻译等。在 Google 公司，互联网网页的搜索索引也是用 Map/Reduce 技术计算生成的。

Map/Reduce 目前已经有基于各种不同计算机编程语言的库实现，其中相当流行的是有 Apache 基金会的用 Java 语言开发的开源的 Hadoop。

1.2.2 资源管理平台

云计算资源规模庞大，服务器数量众多并分布在不同的地点，同时运行着数百种应用，如何有效地管理这些服务器，保证整个系统提供不间断的服务是巨大的挑战。

云计算系统的平台管理技术能够使大量的服务器协同工作，方便进行业务部署和开通，快速发现和恢复系统故障，通过自动化、智能化的手段实现大规模系统的可靠运营。

当前比较流行的云计算平台，主要有思杰的 CloudStack，开源的 Eucalyptus，VMware 公司的 vCloud Director 和开源的 OpenStack 等。除了 VMware 公司的 vCloud Director 没有免费版本和只支持 VMware 自由的虚拟化产品以外，其余几个都提供免费版本，而且支持多个虚拟化产品。Eucalyptus 和 OpenStack 还对亚马逊的 API 保持兼容，基于亚马逊 API 的所有脚本和软件产品都可以轻松地进行私有云部署。

现在，CloudStack 和 OpenStack 正处在激烈的竞争中，两者都希望自己能够成为开源社区云计算平台的事实标准。CloudStack 在成熟度上面明显优于 OpenStack，在培育客户方面也占了先机，但是背后的推手主要是思杰。OpenStack 虽然是后起之秀，但是得到了诸如 IBM、思科、英特尔、惠普和戴尔等大牌厂商的支持。究竟谁是未来的开源云平台老大，鹿死谁手，还未可知。

1.2.3 虚拟化

虚拟化是构建云基础架构不可或缺的关键技术之一。云计算的云端系统，其实质上就是一个大型的分布式系统。虚拟化通过在一个物理平台上虚拟出更多的虚拟平台，而其中的每一个虚拟平台则可以作为独立的终端加入云端的分布式系统。比起直接使用物理平台，虚拟化在资源的有效利用、动态调配和高可靠性方面有着巨大的优势。利用虚拟化，企业不必抛弃现有的基础架构即可构建全新的信息基础架构，从而更加充分地利用原有的 IT 投资。

可以说，虚拟化是云端系统部署必不可少的基础。正是因为虚拟化在云计算中的重要性，本书将集中篇幅以 KVM 为例，为各位读者讲解虚拟化从技术理解到实践部署的方方面面。从下一节开始，本书将进入虚拟化技术的专题。

1.3 虚拟化技术

虚拟化是一个广义的术语，是指计算元件在虚拟的基础上而不是真实的基础上运行，是一个为了简化管理、优化资源的解决方案。

如图 1-1 所示, 我们可以将一般的计算模型抽象成为一定的物理资源和运行于之上的计算元件, 它们之间通过定义的物理资源接口进行交互。随着计算机硬件技术的发展, 物理资源的容量越来越大而价格越来越低, 在既有的计算元件架构下, 物理资源不可避免地产生了闲置和浪费。为了充分利用新的物理资源, 提高效率, 一个比较直接的办法就是更新计算元件以利用更加丰富的物理资源。但是, 人们往往出于对稳定性和兼容性的追求, 并不情愿频繁地对已经存在的计算元件做大幅度的变更。虚拟化技术则是另辟蹊径, 通过引入一个新的虚拟化层, 对下管理真实的物理资源, 对上提供虚拟的系统资源, 从而实现了在扩大硬件容量的同时, 简化软件的重新配置过程。

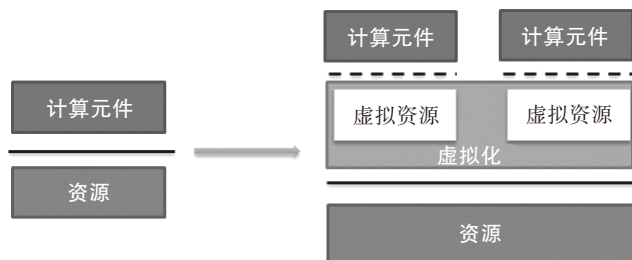


图 1-1 虚拟化

为了表述虚拟化的一般概念, 在图 1-1 中使用了资源这个词。在实际应用中, 资源可以表现为各种各样的形式。例如, 如果把操作系统及其提供的系统调用作为资源, 那么虚拟化就表现为操作系统虚拟化。Linux 容器虚拟化技术就是在同样的一份 Linux 操作系统之上, 虚拟出多个同样的操作系统。再例如, 如果把整个 X86 平台包括处理器、内存和外设作为资源, 那么对应的虚拟化技术就是平台虚拟化, 在同一个 X86 平台上面, 可以虚拟多个 X86 平台, 每个虚拟平台都可以运行自己独立完整的操作系统。这也是我们在本书中将要重点讨论的虚拟化技术。

在 X86 平台虚拟化技术中, 新引入的虚拟化层通常称为虚拟机监控器 (Virtual Machine Monitor, VMM), 也叫做 Hypervisor。虚拟机监控器运行的环境, 也就是真实的物理平台, 称之为宿主机。而虚拟出来的平台通常称为客户机, 里面运行的系统对应地也称为客户机操作系统。

虚拟化技术有很多种实现方式, 比如软件虚拟化和硬件虚拟化, 再比如准虚拟化和全虚拟化。下面将针对每种实现方式做一个简单的介绍。

1.3.1 软件虚拟化和硬件虚拟化

通过前面的简单介绍, 相信读者对虚拟化已经有了一个大致的了解。那么, 什么是软件虚拟化和硬件虚拟化呢? 在回答这个问题之前, 读者需要思考一个问题: 既然虚拟化这么美好, 是不是物理资源都可以被虚拟化?

如图 1-2 所示, 实现虚拟化的重要一步就在于, 虚拟化层必须能够截获计算元件对物理

资源的直接访问，并将其重新定向到虚拟资源池中。根据虚拟化层是通过纯软件的方法，还是利用物理资源提供的机制来实现这种“截获并重定向”，我们可以把虚拟化划分成为软件虚拟化和硬件虚拟化两种。

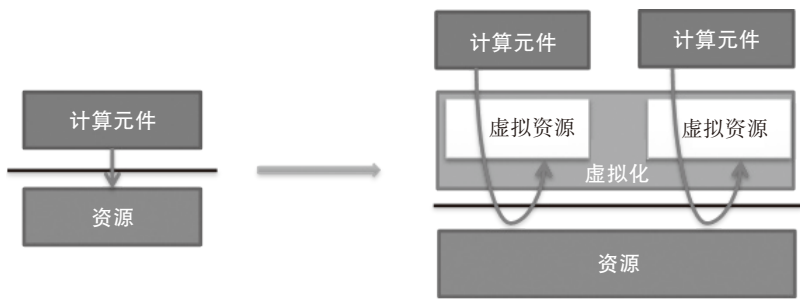


图 1-2 虚拟化的可实现性

1. 虚拟化——软件方案

纯软件虚拟化，顾名思义，就是用纯软件的方法在现有的物理平台上（往往并不支持硬件虚拟化）实现对物理平台访问的截获和模拟。

常见的软件虚拟机例如 QEMU，它是通过纯软件来仿真 X86 平台处理器的取指、解码和执行，客户机的指令并不在物理平台上直接执行。由于所有的指令都是软件模拟的，因此性能往往比较差，但是可以在同一平台上模拟不同架构平台的虚拟机。

VMWare 的软件虚拟化则使用了动态二进制翻译的技术。虚拟机监控机在可控制的范围内，允许客户机的指令在物理平台上直接运行。但是，客户机指令在运行前会被虚拟机监控机扫描，其中突破虚拟机监控机限制的指令会被动态替换为可以在物理平台上直接运行的安全指令，或者替换为对虚拟机监控器的软件调用。这样做的好处是比纯软件模拟性能有大幅的提升，但是也同时失去了跨平台虚拟化的能力。

在纯软件虚拟化解方案中，VMM 在软件套件中的位置是传统意义上操作系统所处的位置，而操作系统的位置是传统意义上应用程序所处的位置，这种转换必然会增加系统的复杂性。软件堆栈的复杂性增加意味着，这些环境难于管理，因而会加大确保系统可靠性和安全性的困难。

2. 虚拟化——硬件方案

硬件虚拟化，简而言之，就是物理平台本身提供了对特殊指令的截获和重定向的硬件支持。甚至，新的硬件会提供额外的资源来帮助软件实现对关键硬件资源的虚拟化，从而提升性能。

以 X86 平台的虚拟化为例，支持虚拟技术的 X86 CPU 带有特别优化过的指令集来控制虚拟过程，通过这些指令集，VMM 会很容易将客户机置于一种受限制的模式下运行，一旦客户机试图访问物理资源，硬件会暂停客户机的运行，将控制权交还给 VMM 处理。VMM 还可以

利用硬件的虚拟化增强机制，将客户机在受限模式下对一些特定资源的访问，完全由硬件重定向到 VMM 指定的虚拟资源，整个过程不需要暂停客户机的运行和 VMM 软件的参与。

由于虚拟化硬件可提供全新的架构，支持操作系统直接在上面运行，无需进行二进制转换，减少了相关的性能开销，极大简化了 VMM 设计，进而使 VMM 能够按通用标准进行编写，性能更加强大。

需要说明的是，硬件虚拟化技术是一套解决方案。完整的情况需要 CPU、主板芯片组、BIOS 和软件的支持，例如 VMM 软件或者某些操作系统本身。即使只是 CPU 支持虚拟化技术，在配合 VMM 软件的情况下，也会比完全不支持虚拟化技术的系统有更好的性能。

鉴于虚拟化的巨大需求和硬件虚拟化产品的广阔前景，Intel 一直都在努力完善和加强自己的硬件虚拟化产品线。自 2005 年末，Intel 便开始在其处理器产品线中推广应用 Intel Virtualization Technology (Intel VT) 虚拟化技术，发布了具有 Intel VT 虚拟化技术的一系列处理器产品，包括桌面的 Pentium 和 Core 系列，还有服务器的 Xeon 至强和 Itanium 安腾。Intel 一直保持在每一代新的处理器架构中优化硬件虚拟化的性能和增加新的虚拟化技术。现在市面上，从桌面的 Core i3/5/7，到服务器端的 E3/5/7/9，几乎全部都支持 Intel VT 技术。可以说，在不远的将来，Intel VT 很可能会成为所有 Intel 处理器的标准配置。

1.3.2 准虚拟化与全虚拟化

1. 准虚拟化 (para-virtualization)

如上一节所述，软件虚拟化可以在缺乏硬件虚拟化支持的平台上完全通过 VMM 软件来实现对各个虚拟机的监控，以保证它们之间彼此独立和隔离。但是付出的代价是软件复杂度的增加，和性能上的损失。减轻这种负担的一种方法就是，改动客户操作系统，使它以为自己运行在虚拟环境下，能够与虚拟机监控机协同工作。这种方法就叫准虚拟化 (para-virtualization)，也叫半虚拟化。本质上，准虚拟化弱化了对虚拟机特殊指令的被动截获要求，将其转化成客户机操作系统的主动通知。但是，准虚拟化需要修改客户机操作系统的源代码来实现主动通知。

Xen 是开源准虚拟化技术的一个例子^[4]。操作系统作为虚拟服务器在 Xen Hypervisor 上运行之前，它必须在内核层面进行某些改变。因此，Xen 适用于 BSD、Linux、Solaris 及其他开源操作系统，但不适合对像 Windows 这些专有的操作系统进行虚拟化处理，因为它们不公开源代码，所以无法修改其内核。

2. 全虚拟化 (full virtualization)

与准虚拟化技术不同，全虚拟化为客户机提供了完整的虚拟 X86 平台，包括处理器、内存和外设，支持运行任何理论上可在真实物理平台上运行的操作系统，为虚拟机的配置提供了最大程度的灵活性。不需要对客户机操作系统做任何修改即可正常运行任何非虚拟化环境中已存在基于 X86 平台的操作系统和软件，是全虚拟化无可比拟的优势。

准虚拟化在诞生之初，运行在缺乏硬件虚拟化支持的平台上，操作系统和虚拟机监控

器的密切配合，实现了针对全虚拟化软件方案的性能优势。这样的性能优势甚至一直保持到 2006 年左右（尽管那时已经出现了第一代硬件虚拟化平台）。但是随着硬件虚拟化技术的迭代演化，运行于 Intel 平台的全虚拟化的性能已经超过了准虚拟化产品，这一点在 64 位的操作系统上表现得更为明显。加之，全虚拟化不需要对客户机操作系统做任何修改的固有优势。可以预言，基于硬件的全虚拟化产品将是未来虚拟化技术的核心。

1.4 KVM 简介

1.4.1 KVM 的历史

KVM 的全称是 Kernel Virtual Machine，翻译成中文就是内核虚拟机。KVM 虚拟机最初是由一个以色列的创业公司 Qumranet 开发的，作为他们的 VDI 产品的虚拟机。为了简化开发，KVM 的开发人员并没有选择从底层开始新写一个 Hypervisor，而是选择了基于 Linux kernel，通过加载新的模块从而使 Linux Kernel 本身变成一个 Hypervisor。2006 年 10 月，在先后完成了基本功能、动态迁移以及主要的性能优化之后，Qumranet 正式对外宣布了 KVM 的诞生。同年 10 月，KVM 模块的源代码被正式接纳进入 Linux Kernel，成为内核源代码的一部分。作为一个功能和成熟度都逊于 Xen 的项目，在这么快的时间内被内核社区接纳，主要原因在于：在虚拟化方兴未艾的当时，内核社区急于将虚拟化的支持包含在内，但是 Xen 取代内核由自身管理系统资源的架构引起了内核开发人员的不满和抵触。

在 2008 年 9 月 4 日，同内核社区保持着很深渊源的著名 Linux 发行版提供商——Redhat 公司出人意料地出资 1 亿 700 百万美金，收购了 Qumranet，从而成为了 KVM 开源项目的新东家。由于此次收购，Redhat 公司有了自己的虚拟机解决方案，于是开始在自己的产品中用 KVM 替换 Xen。2010 年 11 月，Redhat 公司推出了新的企业版 Linux——RHEL 6，在这个发行版中集成了最新的 KVM 虚拟机，而去掉了在 RHEL 5.x 系列中集成的 Xen。

1.4.2 KVM 功能概览

KVM 是基于虚拟化扩展（Intel VT 或 AMD-V）的 X86 硬件，是 Linux 完全原生的全虚拟化解决方案。部分的准虚拟化支持，主要是通过准虚拟网络驱动程序的形式用于 Linux 和 Windows 客户机系统的。KVM 目前设计为通过可加载的内核模块，支持广泛的客户机操作系统，比如 Linux、BSD、Solaris、Windows、Haiku、ReactOS 和 AROS Research Operating System。

在 KVM 架构中，虚拟机实现为常规的 Linux 进程，由标准 Linux 调度程序进行调度。事实上，每个虚拟 CPU 显示为一个常规的 Linux 进程。这使 KVM 能够享受 Linux 内核的所有功能。

需要注意的是，KVM 本身不执行任何模拟，需要用户空间程序通过 /dev/kvm 接口设置一个客户机虚拟服务器的地址空间，向它提供模拟的 I/O，并将它的视频显示映射回宿主的

显示屏。目前这个应用程序就是大名鼎鼎的 QEMU。

图 1-3 显示了 KVM 的基本架构。

下面介绍一下 KVM 的功能特性。

(1) 内存管理

KVM 从 Linux 继承了强大的内存管理功能。一个虚拟机的内存与任何其他 Linux 进程的内存一样进行存储，可以以大页面的形式进行交换以实现更高的性能，也可以以磁盘文件的形式进行共享。NUMA 支持（非一致性内存访问，针对多处理器的内存设计）允许虚拟机有效地访问大量内存。

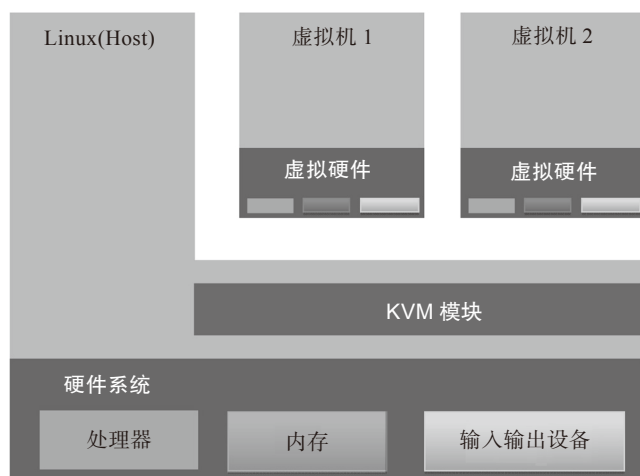


图 1-3 KVM 架构

KVM 支持最新的基于硬件的内存虚拟化功能，支持 Intel 的扩展页表（EPT）和 AMD 的嵌套页表（NPT，也叫“快速虚拟化索引 -RVI”），以实现更低的 CPU 利用率和更高的吞吐量。

内存页面共享通过一项名为内核同页合并（Kernel Same-page Merging, KSM）的内核功能来支持。KSM 扫描每个虚拟机的内存，如果虚拟机拥有相同的内存页面，KSM 将这些页面合并到一个在虚拟机之间共享的页面，仅存储一个副本。如果一个客户机尝试更改这个共享页面，它将得到自己的专用副本。

(2) 存储

KVM 能够使用 Linux 支持的任何存储来存储虚拟机镜像，包括具有 IDE、SCSI 和 SATA 的本地磁盘，网络附加存储（NAS）（包括 NFS 和 SAMBA/CIFS），或者支持 iSCSI 和光纤通道的 SAN。多路径 I/O 可用于改进存储吞吐量和提供冗余。由于 KVM 是 Linux 内核的一部分，它可以利用所有领先存储供应商都支持的一种成熟且可靠的存储基础架构，它的存储堆栈在生产部署方面具有良好的记录。

KVM 还支持全局文件系统（GFS2）等共享文件系统上的虚拟机镜像，以允许虚拟机镜像在多个宿主之间共享或使用逻辑卷共享。磁盘镜像支持按需分配，仅在虚拟机需要时分配存储空间，而不是提前分配整个存储空间，提高存储利用率。KVM 的原生磁盘格式为 QCOW2，它支持快照，允许多级快照、压缩和加密。

（3）设备驱动程序

KVM 支持混合虚拟化，其中准虚拟化的驱动程序安装在客户机操作系统中，允许虚拟机使用优化的 I/O 接口而不使用模拟的设备，从而为网络和块设备提供高性能的 I/O。KVM 准虚拟化的驱动程序使用 IBM 和 Red Hat 联合 Linux 社区开发的 VirtIO 标准，它是一个与虚拟机管理程序独立的、构建设备驱动程序的接口，允许为多个虚拟机管理程序使用一组相同的设备驱动程序，能够实现更出色的虚拟机交互性。

（4）性能和可伸缩性

KVM 也继承了 Linux 的性能和可伸缩性。KVM 虚拟化性能在很多方面（如计算能力、网络带宽等）已经可以达到非虚拟化原生环境的 95% 以上的性能。KVM 的扩展性也非常良好，客户机和宿主机都可以支持非常多的 CPU 数量和非常大量的内存。例如，Redhat 官方文档^[5]就介绍过，RHEL 6.x 系统中的一个 KVM 客户机可以支持 160 个虚拟 CPU 和多达 2TB 的内存，KVM 宿主机支持 4096 个 CPU 核心和多达 64TB 的内存。

1.4.3 KVM 的前景

尽管 KVM 是一个相对较新的虚拟机管理程序，但是诞生不久就被 Linux 社区接纳，成为随 Linux 内核发布的轻量级模块。与 Linux 内核集成，使 KVM 可以直接获益于最新的 Linux 内核开发成果，比如更好的进程调度支持、更广泛物理硬件平台的驱动、更高的代码质量，等等。

作为相对较新的虚拟化方案，KVM 一直没有成熟的工具可用于管理 KVM 服务器和客户机，不过，现在随着 libvirt、virt-manager 等工具和 OpenStack 等云计算平台的逐渐完善，KVM 管理工具在易用性方面的劣势已经逐渐被克服。另外，KVM 仍然可以改进虚拟网络的支持、虚拟存储支持、增强的安全性、高可用性、容错性、电源管理、HPC/ 实时支持、虚拟 CPU 可伸缩性、跨供应商兼容性、科技可移植性等方面，不过，现在 KVM 开发者社区比较活跃，也有不少大公司的工程师参与开发，我们有理由相信很多功能都会在不远的将来得到完善。

1.5 Xen 简介

1.5.1 Xen 的历史

早在 20 世纪 90 年代，伦敦剑桥大学的 Ian Pratt 和 Keir Fraser 在一个叫做 Xenoserver 的研究项目中，开发了 Xen 虚拟机。作为 Xenoserver 的核心，Xen 虚拟机负责管理和分配系

统资源，并提供必要的统计功能。在那个年代，X86 的处理器还不具备对虚拟化技术的硬件支持，所以 Xen 从一开始是作为一个准虚拟化的解决方案出现的。因此，为了支持多个虚拟机，内核必须针对 Xen 做出特殊的修改才可以运行。为了吸引更多开发人员参与，2002 年 Xen 正式被开源。在先后推出了 1.0 和 2.0 版本之后，Xen 开始被诸如 Redhat、Novell 和 Sun 的 Linux 发行版集成，作为其中的虚拟化解决方案。2004 年，Intel 的工程师开始为 Xen 添加硬件虚拟化的支持，从而为即将上市的新款处理器做必需的软件准备。在他们的努力下，2005 年发布的 Xen 3.0，开始正式支持 Intel 的 VT 技术和 IA64 架构，从而 Xen 虚拟机可以运行完全没有修改的操作系统。2007 年 10 月，思杰公司出资 5 亿美金收购了 XenSource，变成了 Xen 虚拟机项目的东家。

与 Xen 在功能开发上的快速进展形成对比的是，Xen 在将它对内核的修改集成进入内核社区方面进展不大。有部分重要的内核开发人员不喜欢 Xen 的架构和实现，多位内核维护人员公开声明不欢迎 Xen。这样的形势一直持续到 2010 年，在基于内核的 PVOPS 对 Xen 做了大量重写之后，内核社区才勉强接纳了 Xen。当然，目前从 Linux 3.0 版本开始的内核主干对 Xen 的支持还是越来越好了。

1.5.2 Xen 功能概览

Xen 是一个直接在系统硬件上运行的虚拟机管理程序。Xen 在系统硬件与虚拟机之间插入一个虚拟化层，将系统硬件转换为一个逻辑计算资源池，Xen 可将其中的资源动态地分配给任何操作系统或应用程序。在虚拟机中运行的操作系统能够与虚拟资源交互，就好像它们是物理资源一样。

图 1-4 显示了一个运行虚拟机的 Xen 系统。

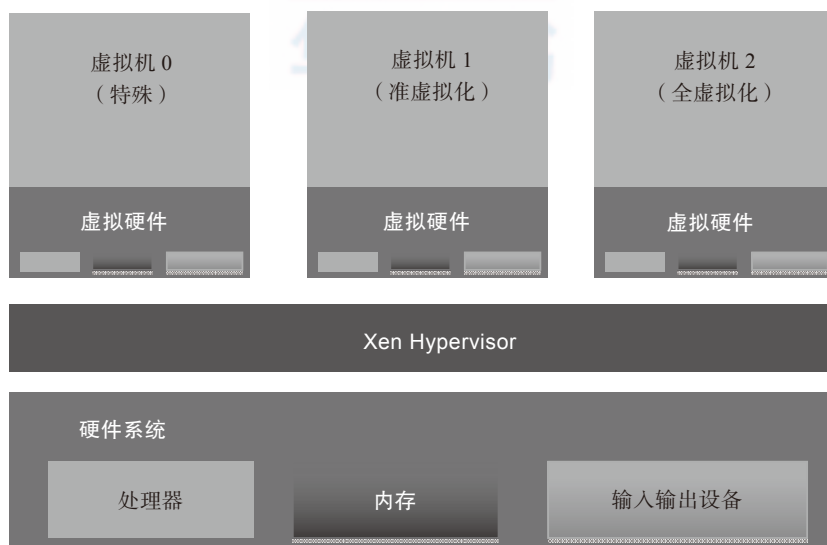


图 1-4 Xen 架构

Xen 被设计成为微内核的实现，其本身只是负责管理处理器和内存资源。Xen 上面运行的所有虚拟机中，0 号虚拟机是特殊的，其中运行的是经过修改的支持准虚拟化的 Linux 操作系统，大部分的输入输出设备都交由这个虚拟机直接控制，而 Xen 本身并不直接控制它们。这样做可以使基于 Xen 的系统可以最大程度地复用 Linux 内核的驱动程序。更广泛地说，Xen 虚拟化方案在 Xen Hypervisor 和 0 号虚拟机的功能上做了聪明的划分，既能够重用大部分 Linux 内核的成熟代码，又可以控制系统之间的隔离型和针对虚拟机更加有效的管理和调度。通常，0 号虚拟机也被视为是 Xen 虚拟化方案的一部分。

Xen 上面运行的虚拟机，既支持准虚拟化，也支持全虚拟化，可以运行几乎所有可以在 X86 物理平台上运行的操作系统。此外，最新的 Xen 还支持 ARM 平台的虚拟化。

下面简单介绍一下 Xen 的功能特性。

Xen 服务器（即思杰公司的 Xen Server 产品）构建于开源的 Xen 虚拟机管理程序之上，结合使用半虚拟化和硬件协助的虚拟化。操作系统与虚拟化平台之间的这种协作支持开发一个较简单的虚拟机管理程序来提供高度优化的性能。

Xen 提供了复杂的工作负载平衡功能，可捕获 CPU、内存、磁盘 I/O 和网络 I/O 数据，它提供了两种优化模式：一种针对性能，另一种针对密度。

Xen 服务器利用一种名为 Citrix Storage Link 的独特的存储集成功能。使用 Citrix Storage Link，系统管理员可直接利用来自 HP、Dell Equal Logic、NetApp、EMC 等公司的存储产品。

Xen 服务器包含多核处理器支持、实时迁移、物理服务器到虚拟机转换（P2V）和虚拟到虚拟转换（V2V）工具、集中化的多服务器管理、实时性能监控，以及对 Windows 和 Linux 客户机的良好性能。

1.5.3 Xen 的前景

Xen 作为一个开发最早的虚拟化方案，对各种虚拟化功能的支持相对完善。Xen 虚拟机监控程序，是一个专门为虚拟机开发的微内核，所以其资源管理和调度策略完全是针对虚拟机的特性而开发的。作为一个独立维护的微内核，Xen 的功能明确，开发社区构成比较简单，所以更容易接纳专门针对虚拟化所做的功能和优化。

Xen 比较难于配置和使用，部署会占用相对较大的空间，而且非常依赖于 0 号虚拟机中的 Linux 操作系统。Xen 微内核直接运行于真实物理硬件之上，开发和调试都比基于操作系统的虚拟化困难。Xen 最大的困难在于 Linux 内核社区的抵制，导致 Xen 相关的内核改动一直不能顺利进入内核源代码，从而无法及时得到内核最新开发成果的支持，这与 KVM 形成了鲜明的对比。

1.6 其他虚拟化方案简介

1.4 节和 1.5 节分别介绍了两个最有名的开源虚拟机，开放源代码的好处在于可以有人数

庞大的开发社区作为支撑，同时源代码公开也有利于人们研究和学习虚拟机的具体实现。但是，把虚拟机作为商品出售给终端用户，就需要一些商业化的虚拟机解决方案。下面简单介绍常见的一些商业化的虚拟机。

1.6.1 VMware

VMware 公司创办于 1998 年，从公司的名字就可以看出，这是一家专注于提供虚拟化解决方案的公司。VMware 公司很早就预见到了虚拟化在未来数据中心中的核心地位，有针对性地开发虚拟化软件，从而抓住了 21 世纪初虚拟化兴起的大潮，成为了虚拟化业界的标杆。VMware 公司从创建至今，一直占据着虚拟化软件市场的最大份额，是毫无争议的龙头老大。Vmware 公司作为最成熟的商业虚拟化软件提供商，其产品线是业界覆盖范围最广的，下面会对 VMware 的主要产品进行简单的介绍。

(1) VMware Workstation

VMware Workstation 是 VMware 公司销售的运行于台式机和 workstation 上的虚拟化软件，也是 VMware 公司第一个面市的产品（1999 年 5 月）。该产品最早采用了 VMware 在业界知名的二进制翻译技术，在 x86 CPU 硬件虚拟化技术还未出现之前，为客户提供了纯粹的基于软件的全虚拟化解决方案。作为最初的拳头产品，VMware 公司投入了大量的资源对二进制翻译进行优化，其二进制翻译技术带来的虚拟化性能甚至超过第一代的 CPU 硬件虚拟化产品。该产品如同 KVM，是“类型二”虚拟机^[6]，需要在宿主操作系统之上运行。

(2) VMware ESX Server

ESX 服务器（一种能直接在硬件上运行的企业级的虚拟平台），虚拟的 SMP，它能让一个虚拟机同时使用四个物理处理器，和 VMFS 一样，它能使多个 ESX 服务器分享块存储器。该公司还提供一虚拟中心来控制和管理虚拟化的 IT 环境：VMotion 让用户可以移动虚拟机器；DRS 从物理处理器创造资源工具；HA 提供从硬件故障自动恢复功能；综合备份可使 LAN-free 自动备份虚拟机器；VMotion 存储器可允许虚拟机磁盘自由移动；更新管理器自动更新修补程序和更新管理；能力规划能使 VMware 的服务供应商执行能力评估；转换器把本地和远程物理机器转换到虚拟机器；实验室管理可自动化安装、捕捉、存储和共享多机软件配置；ACE 允许桌面系统管理员对虚拟机应用统一的企业级 IT 安全策略，以防止不可控台式电脑带来的风险。虚拟桌面基础设施可主导个人台式电脑在虚拟机运行的中央管理器；虚拟桌面管理，它是联系用户到数据库中的虚拟电脑的桌面管理服务器；VMware 生命管理周期可通过虚拟环境提供控制权。

1.6.2 VirtualBox

Oracle VirtualBox 是由德国 InnoTek 软件公司出品的虚拟机器软件，现在由甲骨文公司进行开发，是甲骨文公司 xVM 虚拟化平台技术的一部份。它提供使用者在 32 位或 64 位的 Windows、Solaris 及 Linux 操作系统上虚拟其他 X86 的操作系统。使用者可以在 VirtualBox

上安装并执行 Solaris、Windows、DOS、Linux、OS/2 Warp、OpenBSD 及 FreeBSD 等系统作为客户端操作系统。最新的 VirtualBox 还支持运行 Android4.0 系统。

与同性质的 VMware 及 Virtual PC 比较下, VirtualBox 独到之处包括远端桌面协定 (RDP)、iSCSI 及 USB 的支援, VirtualBox 在客户机操作系统上已可以支援 USB 2.0 的硬件装置。此外, VirtualBox 还支持在 32 位宿主操作系统上运行 64 位的客户机操作系统。

VirtualBox 既支持纯软件虚拟化, 也支持 Intel VT-x 与 AMD AMD-V 硬件虚拟化技术。为了方便其他虚拟机用户向 VirtualBox 的迁移, VirtualBox 可以读写 VMware VMDK 格式与 VirtualPC VHD 格式的虚拟磁盘文件。

1.6.3 Hyper-V

Hyper-V 是微软提出的一种系统管理程序虚拟化技术。Hyper-V 设计的目的是为广泛的用户提供更为熟悉及成本效益更高的虚拟化基础设施软件, 这样可以降低运作成本、提高硬件利用率、优化基础设施并提高服务器的可用性。

Hyper-V 的设计借鉴了 Xen, 采用微内核的架构, 兼顾了安全性和性能的要求。Hyper-V 底层的 Hypervisor 运行在最高的特权级别下, 微软将其称为 ring -1 (而 Intel 也将其称为 root mode), 而虚拟机的操作系统内核和驱动运行在 ring 0, 应用程序运行在 ring 3。

Hyper-V 采用基于 VMBus 的高速内存总线架构, 来自虚拟机的硬件请求 (显卡、鼠标、磁盘、网络), 可以直接经过 VSC, 通过 VMBus 总线发送到根分区的 VSP, VSP 调用对应的设备驱动, 直接访问硬件, 中间不需要 Hypervisor 的帮助。

从架构上讲, Hyper-V 只有“硬件—Hyper-V—虚拟机”三层, 本身非常小巧, 代码简单, 且不包含任何第三方驱动, 所以安全可靠、执行效率高, 能充分利用硬件资源, 使虚拟机系统性能更接近真实系统性能。

1.7 本章小结

本章从云计算的概念和发展历史开始, 逐渐为读者理清了云计算和虚拟化技术之间的联系: 虚拟化技术是实现云计算的基础。

本章还简单介绍了虚拟化技术的本质, 以及常见虚拟化技术的分类。作为重点, 以 KVM 和 Xen 为例, 重点介绍了这两个开源社区最有名的虚拟机。这些内容为第 2 章继续展开深入了解 KVM 的架构做了必要的知识铺垫。

最后, 对常见的商业化虚拟解决方案做了简单的描述, 供有兴趣的读者作为参考。

1.8 本章注释

[1] SETI@home 科学实验, 可参考其网站主页: <http://setiathome.berkeley.edu/>

- [2] Google Docs 服务从 2012 年 4 月开始已经被合并到新推出的 Google Drive 服务之中。Google Drive 是 Google 提供的文件存储和同步服务，其网站地址为：<https://drive.google.com/>
- [3] 维基百科中对云计算的定义也在不断变化中，见：http://en.wikipedia.org/wiki/Cloud_computing
- [4] 准确地讲，Xen 最初是从准虚拟化发展起来的，但目前也能很好地利用硬件虚拟化技术实现全虚拟化，可以创建未经修改过的客户机。
- [5] Redhat 关于 Redhat Linux 中对于包括 KVM 在内的 Hypervisor 的限制，官方文档的网页为：<http://www.redhat.com/resourcelibrary/articles/virtualization-limits-rhel-hypervisors>
- [6] 关于“类型一”和“类型二”虚拟机分类的标准，可参考 2.3 节中的介绍。



第2章 KVM原理简介

2.1 Linux 操作系统简介

Linux 是一种自由和开放源代码的类 UNIX 操作系统内核。目前存在着许多不同的 Linux 发布版，可安装在各种各样的电脑硬件设备上，从手机、平板电脑、路由器和影音游戏控制台，到台式机、大型机和超级电脑。Linux 是一个领先的操作系统内核，世界上运算最快的 10 台超级电脑运行的都是基于 Linux 内核的操作系统。

Linux 操作系统也是自由软件和开放源代码发展中最著名的例子。只要遵循 GNU 通用公共许可证，任何人和机构都可以自由地使用 Linux 的所有底层源代码，也可以自由地修改和再发布。严格来讲，Linux 这个词本身只表示 Linux 内核，但实际上人们已经习惯了用 Linux 来表示整个基于 Linux 内核且使用 GNU 工程各种工具构建的操作系统（也被称为 GNU/Linux）。在通常情况下，Linux 被打包成供台式机和服务器使用的 Linux 发布版本。

1. Linux 的历史

1991 年，芬兰人 Linus Torvalds 在赫尔辛基大学上学，对操作系统很好奇，热衷于使用 MINIX（一种教学使用的廉价 UNIX）。但是，Linus 对 MINIX 只允许在教育上使用很不满（不允许任何商业使用），于是开始写他自己的操作系统，这就是后来的 Linux 内核。

Linus 开始在 MINIX 上开发 Linux 内核，为 MINIX 写的软件也可以在 Linux 内核上使用。后来 Linux 成熟了，可以在 Linux 上面开发自身的 Linux 内核。为了让 Linux 可以在商业上使用，Linus Torvalds 决定在 Linux 中改变 MINIX 的协议（这个协议会限制商业使用），使用 GNU GPL 协议来代替。开发者致力于融合 GNU 元素到 Linux 中，做出一个有完整功能的、自由的操作系统。

Linux 渐渐得以广泛使用，并赢得了众多的用户。Linux 成功的一个重要因素是，它很快吸引了很多开发者，这些开发人员对其代码进行了修改和完善。直至今日，Linux 仍然是一个基于因特网的协作开发项目。Linus 作为 Linux 之父，现在仍是一位内核维护者，但是开发工作主要是由一个结构松散的内核社区上的开发人员共同完成的。随着 Linux 的商业前景逐渐被看好，计算机行业巨头（如 IBM、Intel 等）都投入自己的开发人员与内核社区一起开发和改进 Linux。

2. 模块化设计的 Linux

操作系统内核设计一直分为两个阵营：微内核和单内核。

单内核是两大阵营中一种较为简单的设计，指的是整个内核从整体上作为一个单独的大过程来实现，并且同时运行在一个单独的地址空间内。所有的内核服务都在这样一个大的内核空间运行，内核之间的通信可以简单地实现为函数调用。这样的设计具有简单高效的特点。但是，如果使用单内核设计，每次对内核作出修改（比如增加或者删除驱动程序），都必须重新编译源代码，生成新的二进制文件，造成了使用和部署上的麻烦。

微内核并不是作为一个单独的大过程来实现的，相反，内核的功能被划分成为多个独立的过程，每一个过程叫做一个服务器。多个服务器程序都运行在自己的地址空间，只有少量核心的服务器运行在特权模式下，服务器之间的通信采用了进程间通信机制。独立的服务器进程提高系统的健壮性，但是进程间通信由于涉及内核空间 and 用户空间的上下文切换，其开销远比函数调用大得多。

Linux 采用了实用主义的设计。为了满足性能要求，Linux 内核被设计成单内核。但是，Linux 内核同时借鉴了微内核的精华：模块化设计以及动态装载内核模块的能力。除了诸如进程切换、内存管理等核心的内核功能，将大部分内核功能作为单独的内核模块设计并实现。这些内核模块编译好后以单独的二进制文件的形式存在，内核在运行过程中，按照需求，动态地加载并链接进入内核空间运行。不使用的模块还可以在运行的过程中动态卸载。这样的设计，既保证了内核的性能，也改进了传统单内核设计的灵活性。

Linux 内核的开放源代码特性及模块化的设计，使其成为很多开发人员和计算机专业学生研究操作系统内核的典型范例。开发人员可以充分利用 Linux 内核已经实现的成熟功能，在其基础上设计、实现自己的内核模块，以此来扩充内核的功能，满足自己的要求。例如，KVM 就是以内核模块的形式存在，为 Linux 内核增加了虚拟化的功能。

2.2 虚拟化模型

通过 1.3 节的阐述，大家应该已经建立了虚拟化的基本概念。在这一节，将针对当下最流行的 X86 平台的虚拟化做进一步的详述。这部分的知识也是理解 KVM 的必要准备。从本节开始，使用“虚拟化”一词时，如果没有特殊说明，都是指 X86 平台的虚拟化。

图 2-1 为大家展示了基本的虚拟化模型。

处于底层是整个物理系统，也就是我们平常看得见、摸得着的系统硬件，主要包括处理器、内存和输入输出设备（这一点相信有主机 DIY 经验的读者是非常熟悉的）。

在物理系统之上，与以往熟悉的操作系统模型不同，运行的是虚拟机监控器（缩写为 VMM 或 Hypervisor）。虚拟机监控器的主要职能是：管理真实的物理硬件平台，并为每个虚拟客户机提供对应的虚拟硬件平台。

图 2-1 中绘制了 3 个虚拟机的实例，每个虚拟机看起来就像是一个小的但是完整的计算机系统，具有自己的“系统硬件”，包括自己的处理器、内存和输入输出设备。在这个计算机系统上，运行着虚拟机自己的操作系统，例如 Linux 和 Windows。

有读者或许会问，既然每一个虚拟机看起来就是一个小的“真实”计算机系统，那里面可以运行自己的虚拟机监控器吗？答案是肯定的。这种情况一般称为“嵌套虚拟化”。KVM支持嵌套虚拟化技术，只是嵌套虚拟化的实现还远远没有达到很稳定和成熟的状态。嵌套虚拟化并不是本章要讲述的内容。



图 2-1 虚拟化模型

一个 X86 平台的核心是其中的处理器，处理器运行程序代码，访问内存和输入输出设备。所以，X86 平台虚拟化技术的核心部分是处理器的虚拟化。只要处理器虚拟化技术支持“截获并重定向”，内存和输入输出设备的虚拟化都可以基于处理器虚拟化技术之上实现。在处理器虚拟化技术的基础上，为了增强虚拟机的性能，内存虚拟化和 IO 虚拟化的新技术也不断被加入到 X86 平台虚拟化技术中。X86 平台虚拟化技术从开始单一的处理器的虚拟化开始，逐步牵涉芯片组、网卡、存储设备以及 GPU 的虚拟化。在 2.6 节，我们将以 Intel 硬件平台为例，详细阐述 X86 平台的硬件虚拟化相关技术的演进。

2.3 KVM 架构

了解了虚拟化的基本模型之后，接下来，我们来看一下 KVM 的具体架构。

从虚拟机的基本架构上来区分，虚拟机一般分为两种，我们称之为类型一和类型二。

其中，“类型一”虚拟机是在系统上电之后首先加载运行虚拟机监控程序，而传统的操作系统则是运行在其创建的虚拟机中。类型一的虚拟机监控程序，从某种意义上说，可以视为一个特别为虚拟机而优化裁剪的操作系统内核。因为，虚拟机监控程序作为运行在底层的软件层，必须实现诸如系统的初始化、物理资源的管理等操作系统的职能；它对虚拟机的创建、调度和管理，与操作系统对进程的创建、调度和管理有共通之处。这一类型的虚拟机监控程序一般会提供一个具有一定特权的特殊虚拟机，由这个特殊虚拟机来运行需要提供给

用户日常操作和管理使用的操作系统环境。著名的开源虚拟化软件 Xen、商业软件 VMware ESX/ESXi 和微软的 Hyper-V 就是“类型一”虚拟机的代表。

与“类型一”虚拟机的方式不同，“类型二”虚拟机监控程序，在系统上电之后仍然运行一般意义上的操作系统（也就是俗称的宿主机操作系统），虚拟机监控程序作为特殊的应用程序，可以视作操作系统功能的扩展。对于“类型二”的虚拟机来说，其最大的优势在于可以充分利用现有的操作系统。因为虚拟机监控程序通常不必自己实现物理资源的管理和调度算法，所以实现起来比较简洁。但是，正所谓“成也萧何，败也萧何”，这一类型的虚拟机监控程序既然依赖操作系统来实现管理和调度，就同样也会受到宿主操作系统的一些限制。例如，通常无法仅仅为了虚拟机的优化，而对操作系统作出修改。本书的主角 KVM 就是属于“类型二”虚拟机，另外，VMware Workstation、VirtualBox 也是属于“类型二”虚拟机。

了解了基本的虚拟机架构之后，让我们来看一下如图 2-2 所示的 KVM 的基本架构。显而易见，KVM 是一个基于宿主操作系统的类型二虚拟机。在这里，我们再一次看到了实用至上的 Linux 设计哲学，既然类型二的虚拟机是最简洁和容易实现的虚拟机监控程序，那么就通过内核模块的形式实现出来就好。其他的部分则尽可能充分利用 Linux 内核的既有实现，最大限度地重用代码。

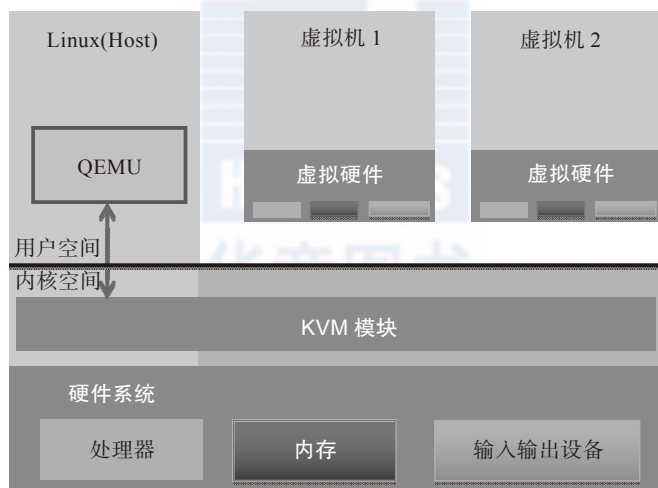


图 2-2 KVM 架构

在图 2-2 中，左侧部分是一个标准的 Linux 操作系统，可以是 RHEL、Fedora、Ubuntu 等。KVM 内核模块在运行时按需加载进入内核空间运行。KVM 本身不执行任何设备模拟，需要用户空间程序 QEMU 通过 /dev/kvm 接口设置一个虚拟客户机的地址空间，向它提供模拟的 I/O 设备，并将它的视频显示映射回宿主机的显示屏。

下面两小节将专门对 KVM 模块和 QEMU 这两个组成部分做简单的介绍。

2.4 KVM 模块

KVM 模块是 KVM 虚拟机的核心部分。其主要功能是初始化 CPU 硬件，打开虚拟化模式，然后将虚拟客户机运行在虚拟机模式下，并对虚拟客户机的运行提供一定的支持。

为了软件的简洁和性能，KVM 仅支持硬件虚拟化。自然而然，打开并初始化系统硬件以支持虚拟机的运行，是 KVM 模块的职责所在。以 KVM 在 Intel 公司的 CPU 上运行为例，在被内核加载的时候，KVM 模块会先初始化内部的数据结构；做好准备之后，KVM 模块检测系统当前的 CPU，然后打开 CPU 控制寄存器 CR4 中的虚拟化模式开关，并通过执行 VMXON 指令将宿主操作系统（包括 KVM 模块本身）置于虚拟化模式中的根模式；最后，KVM 模块创建特殊设备文件 `/dev/kvm` 并等待来自用户空间的命令。接下来虚拟机的创建和运行将是一个用户空间的应用程序（QEMU）和 KVM 模块相互配合的过程。

KVM 模块与用户空间 QEMU 的通信接口主要是一系列针对特殊设备文件的 IOCTL 调用。

如上所述，KVM 模块加载之初，只存在 `/dev/kvm` 文件，而针对该文件的最重要的 IOCTL 调用就是“创建虚拟机”。在这里，“创建虚拟机”可以理解成 KVM 为了某个特定的虚拟客户机（用户空间程序创建并初始化）创建对应的内核数据结构。同时，KVM 还会返回一个文件句柄来代表所创建的虚拟机。针对该文件句柄的 IOCTL 调用可以对虚拟机做相应的管理，比如创建用户空间虚拟地址和客户机物理地址及真实内存物理地址的映射关系，再比如创建多个可供运行的虚拟处理器（vCPU）。同样，KVM 模块会为每一个创建出来的虚拟处理器生成对应的文件句柄，对虚拟处理器相应的文件句柄进行相应的 IOCTL 调用，就可以对虚拟处理器进行管理。

针对虚拟处理器的最重要的 IOCTL 调用就是“执行虚拟处理器”。通过它，用户空间准备好的虚拟机在 KVM 模块的支持下，被置于虚拟化模式中的非根模式下，开始执行二进制指令。在非根模式下，所有敏感的二进制指令都会被处理器捕捉到，处理器在保存现场之后自动切换到根模式，由 KVM 决定如何进一步处理（要么由 KVM 模块直接处理，要么返回用户空间交由用户空间程序处理）。

除了处理器的虚拟化，内存虚拟化也是由 KVM 模块实现的。

实际上，内存虚拟化往往是一个虚拟机实现中代码量最大、实现最复杂的部分（至少，在硬件支持二维地址翻译之前是这样的）。众所周知，处理器中的内存管理单元（MMU）是通过页表的形式将程序运行的虚拟地址转换成为物理内存地址。在虚拟机模式下，内存管理单元的页表则必须在一次查询的时候完成两次地址转换。这是因为，除了要将客户机程序的虚拟地址转换为客户机物理地址以外，还必须将客户机物理地址转换成为真实物理地址。KVM 模块开始使用了影子页表的技术来解决这个问题：在客户机运行时候，处理器真正使用的页表并不是客户机操作系统维护的页表，而是 KVM 模块根据这个页表维护的另外一套影子页表。影子页表的机制比较复杂，感兴趣的读者可以自行翻阅相关材料，这里不再展开详述。

影子页表实现复杂，而且有时候开销很大。为了解决这个问题，新的处理器在硬件上做了增强（Intel 的 EPT 技术）。通过引入第二级页表来描述客户机虚拟地址和真实物理地址的转换，硬件可以自动进行两级转换生成正确的内存访问地址。KVM 模块将其称为二维分页机制。

处理器对设备的访问主要是通过 IO 指令和 MMIO，其中 IO 指令会被处理器直接截获，MMIO 会通过配置内存虚拟化来捕捉。但是，外设的模拟一般并不由 KVM 模块负责。一般来说，只有对性能要求比较高的虚拟设备才会由 KVM 内核模块来直接负责，比如虚拟中断控制器和虚拟时钟，这样可以大量减少处理器的模式切换的开销。大部分的输入输出设备还是会交给下一节将要介绍的用户态程序 QEMU 来负责。

2.5 QEMU 设备模型

QEMU 本身并不是 KVM 的一部分，其自身就是一个著名的开源虚拟机软件。与 KVM 不同，QEMU 虚拟机是一个纯软件的实现，所以性能低下。但是，其优点是在支持 QEMU 本身编译运行的平台上就可以实现虚拟机的功能，甚至虚拟机可以与宿主机并不是同一个架构。作为一个存在已久的虚拟机，QEMU 的代码中有整套的虚拟机实现，包括处理器虚拟化、内存虚拟化，以及 KVM 使用到的虚拟设备模拟（比如网卡、显卡、存储控制器和硬盘等）。

为了简化开发和代码重用，KVM 在 QEMU 的基础上进行了修改。虚拟机运行期间，QEMU 会通过 KVM 模块提供的系统调用进入内核，由 KVM 模块负责将虚拟机置于处理器的特殊模式运行。遇到虚拟机进行输入输出操作，KVM 模块会从上次的系统调用出口处返回 QEMU，由 QEMU 来负责解析和模拟这些设备。

从 QEMU 角度来看，也可以说 QEMU 使用了 KVM 模块的虚拟化功能，为自己的虚拟机提供硬件虚拟化的加速，从而极大地提高了虚拟机的性能。除此之外，虚拟机的配置和创建，虚拟机运行依赖的虚拟设备，虚拟机运行时的用户操作环境和交互，以及一些针对虚拟机的特殊技术（诸如动态迁移），都是由 QEMU 自己实现的。

从 QEMU 和 KVM 模块之间的关系可以看出，这是典型的开源社区在代码共用和开发项目共用上面的合作。诚然，QEMU 可以选择其他的虚拟机或技术来加速，比如 Xen 或者 KQEMU；KVM 也可以选择其他的用户空间程序作为虚拟机实现，只要它按照 KVM 提供的 API 来设计。但是在现实中，QEMU 与 KVM 两者的结合是最成熟的选择，这对一个新开发和后起的项目（KVM）来说，无疑多了一份未来成功的保障。

2.6 Intel 虚拟化技术

Intel 虚拟化技术其实是一系列硬件技术的集合，虚拟机监控机软件通过选择利用各项技术，从而提高虚拟化软件的性能或者实现各种不同的功能。

如图 2-3 所示，Intel 虚拟化技术其实可以大致分为三类：第一类是处理器相关的，称为 VT-x，是实现处理器虚拟化的硬件扩展，这也是硬件虚拟化的基础；第二类是芯片组相关的，成为 VT-d，是从芯片组的层面为虚拟化提供必要支持，通过它，可以实现诸如直接分配物理设备给客户机的功能；第三类是输入输出设备相关的，主要目的是通过定义新的输入输出协议，使新一代的输入输出设备可以更好地支持虚拟化环境下的工作，比如 Intel 网卡自有的 VMDq 技术和 PCI 组织定义的单根设备虚拟化协议（SR-IOV）。

Vector 3: IO Device Focus

Vector 2: Chipset Focus

Vector 1: Processor Focus

VMM Software Evolution

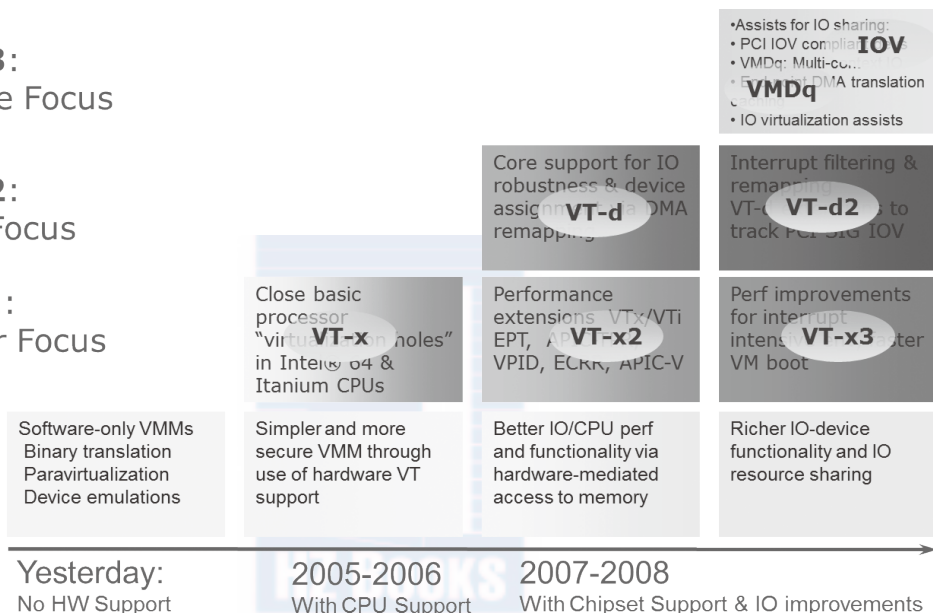


图 2-3 Intel 虚拟化技术进化蓝图

从图 2-3 中可以看出，硬件虚拟化技术发展到今天，已经从最初的处理器虚拟化扩展变成了各种先进技术的集合。而且，Intel 公司还在继续为这个集合添加新的成员。同时 Intel 公司的软件开发工程师也一直活跃在开源虚拟机社区，为 KVM 即时添加最新的硬件支持。对这方面感兴趣的读者，可以自己去下载 KVM 和 QEMU 的源代码，从而进行更深入的理解和应用。

2.7 本章小结

本章首先简单介绍了 Linux 操作系统和内核，接着引入了 X86 平台虚拟化的基本模型。在此基础之上，本章重点阐述了 KVM 的技术架构及组成要素，并对每一个组成要素做了简单介绍。最后，介绍了 KVM 实现所依赖的硬件虚拟化技术进化蓝图。

本章主要是为读者介绍 KVM 的架构、基本组成部分和功能，使读者对 KVM 建立一个初步的技术印象，以便读者可以更容易理解后面章节中关于 KVM 虚拟化的具体内容，另外也可以让在读者阅读 QEMU/KVM 等代码时有一个整体的概念。

第3章 构建KVM环境

在通过第2章了解KVM的基本原理之后，你是否迫不及待地想实践一下如何使用KVM来构建自己的虚拟化环境呢？那么，请仔细阅读本章吧！

本章将介绍如何通过整套的流程与方法来构建KVM环境，其中包括：硬件系统的配置、宿主机（Host）操作系统的安装、KVM的编译与安装、qemu-kvm的编译与安装、客户机（Guest）的安装，直到最后启动属于你的第一个KVM客户机。

3.1 硬件系统的配置

我们知道，KVM从诞生伊始就是需要硬件虚拟化扩展的支持，所以这里需要特别讲解一下硬件系统的配置。

KVM最初的开发是基于x86和x86-64处理器架构上的Linux系统进行的，目前，KVM被移植到多种不同处理器架构之上，包括：Intel和HP的IA64（安腾）架构、AIM联盟（Apple-IBM-Motorola）的PowerPC架构、IBM的S/390架构、ARM架构（2012年开始^[1]）。其中，在X86-64上面的功能支持最完善（主要原因是Intel/AMD的x86-64架构在桌面和服务器的市场的主导地位及其架构的开放性，以及它的开发者众多），本书也采用基于Intel x86-64架构的处理器作为基本的硬件环境^[2]。

在x86-64架构的处理器中，KVM必需的硬件虚拟化扩展分别为：Intel的虚拟化技术（Intel VT）和AMD的AMD-V技术。其中，Intel在2005年11月发布的奔腾四处理器（型号：662和672）第一次正式支持VT技术（Virtualization Technology），之后不久的2006年5月AMD也发布了支持AMD-V的处理器。现在比较流行的针对服务器和桌面的Intel处理器多数都是支持VT技术的，本节着重讲述英特尔的VT技术相关的硬件设置。

首先处理器（CPU）要在硬件上支持VT技术，还要在BIOS中将其功能打开，KVM才能使用到。目前，多数流行的服务器和部分桌面处理器的BIOS都默认将VT打开了。

一般在BIOS中，VT的选项通过“Advanced → Processor Configuration”来查看和设置，它的标识通常为“Intel(R) Virtualization Technology”或“Intel VT”等类似的文字说明。

除了支持必需的处理器虚拟化扩展以外，如果服务器芯片还支持VT-d，就建议在BIOS中将其打开，因为后面一些相对高级的设备的直接分配功能会需要硬件VT-d技术的支持。VT-d（Virtualization Technology for Directed I/O）是对设备I/O的虚拟化硬件支持，在BIOS中的位置可能为“Advanced → Processor Configuration”或“Advanced → System Agent (SA) Configuration”，

它一般在 BIOS 中的标志一般为 “Intel(R) VTfor Directed I/O” 或 “Intel VT-d” 等。

下面以一台 Intel Romley-EP 平台的服务器为例来说明在 BIOS 中的设置。

1) BIOS 中的 Advanced 选项，如图 3-1 所示。

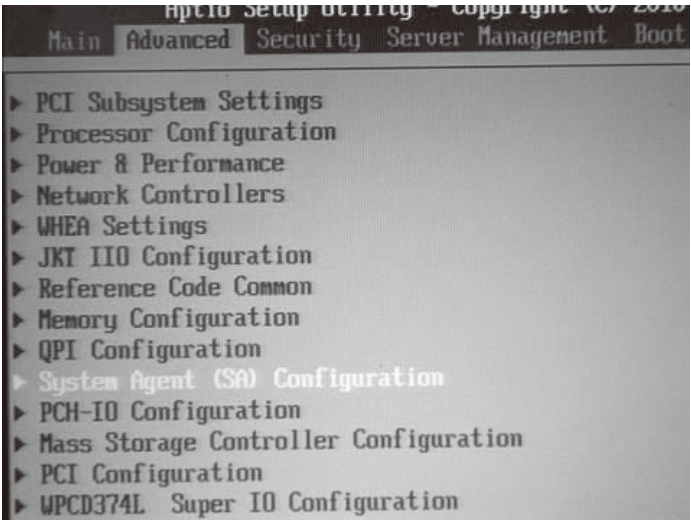


图 3-1 BIOS 中的 Advanced 选项

2) BIOS 中的 VT 和 VT-d 选项，如图 3-2 所示。

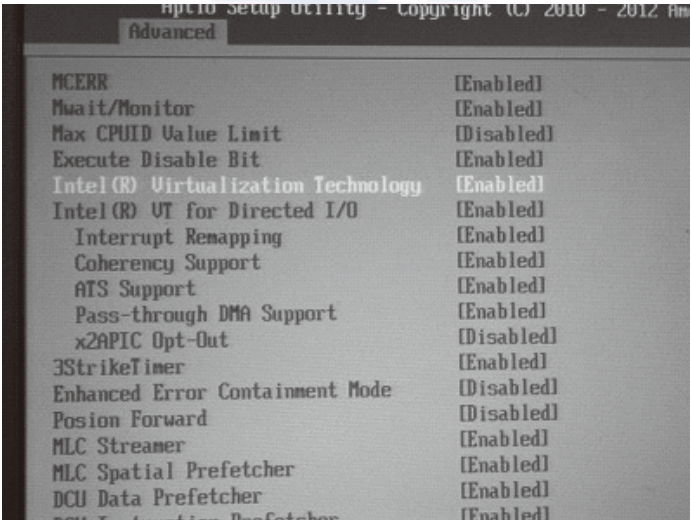


图 3-2 BIOS 中 Enabled 的 VT 与 VT-d 选项

对于不同平台或不同厂商的 BIOS，VT 和 VT-d 等设置的位置可能是不一样的，需要根据实际的硬件情况和 BIOS 中的选项来灵活设置。

设置好了 VT 和 VT-d 的相关选项，保存 BIOS 的设置并退出，系统重启后生效。在

Linux 系统中，可以通过检查 `/proc/cpuinfo` 文件中的 CPU 特性标志（flags）来查看 CPU 目前是否支持硬件虚拟化。在 x86 和 x86-64 平台中，Intel 系列 CPU 支持虚拟化的标志为“vmx”，AMD 系列 CPU 的标志为“svm”，所以可以用如下命令行查看“vmx”或者“svm”标志。

```
[root@jay-linux ~]# grep -E '(vmx|svm)' /proc/cpuinfo
flags              : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall
nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl
xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl
vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic popcnt
tsc_deadline_timer aes xsave avx lahf_lm ida arat epb xsaveopt pln pts
dts tpr_shadow vnmi flexpriority ept vpid
```

<!-- 此处省略多行其余 CPU 或 core 的 flags 输出信息 -->

3.2 安装宿主机 Linux 系统

KVM 是基于内核的虚拟化技术，要运行 KVM 虚拟化环境，安装一个 Linux 操作系统的宿主机（Host）是必需的。由于 Redhat 公司是目前对 KVM 项目投入最多的企业之一，从 RHEL6 开始，其系统自带的虚拟化方案就采用了 KVM，而且 RHEL（Red Hat Enterprise Linux）也是最流行的企业级 Linux 发行版之一，所以本节选用 RHEL 来讲解 Linux 系统的安装步骤和过程，并且本章后面的编译和运行都是在这个系统上进行的。

当然，KVM 作为流行的开源虚拟机之一，它可以在绝大多数流行的 Linux 系统上编译和运行，所以依然可以选择 RHEL 之外的其他 Linux 发行版，CentOS、Fedora、Ubuntu、Debian、OpenSuse 等系统都是不错的选择。

本节内容基于目前最新的 RHEL 版本——RHEL6.3 Server 版的系统来简单介绍，普通 Linux 安装的基本过程这里就不再详细描述，这里主要说明安装过程中一些值得注意的地方。

在选择哪种类型的服务器时，选择“Software Development Workstation”即可（如图 3-3 所示），然后选中当前页面的“Customize now”，单击“Next”按钮进入下一步去选择具体需要安装的组件并设置所需要的各个 RPM 包。

在选择了“Software Development Workstation”之后，在选择具体组件的界面就可以看到默认已经选择了很多的组件（如图 3-4 所示），这里主要需要检查一下 Development 这个选项中默认已经勾选了的很多的开发组件。其中，最好选中 Development 选项中的 Development tools 和 Additional Development 这两个组件，因为在本书的 KVM 编译过程中以及其他实验中可能会用到，其中包括一些比较重要的软件包，比如：gcc、git、make 等（一般被默认选中）。另外可以单击下方的“Optional packages”按钮根据需要选择一些可选的软件包。在图 3-4 的 Virtualization 选项中，我们可以先不选中其中的任何组件，因为在本章中会自己编译 KVM 和 qemu-kvm，而在第 7 章介绍发行版中的 KVM 时，一般会安装 Virtualization 组件并使用发行版中自带的 KVM Virtualization 功能。

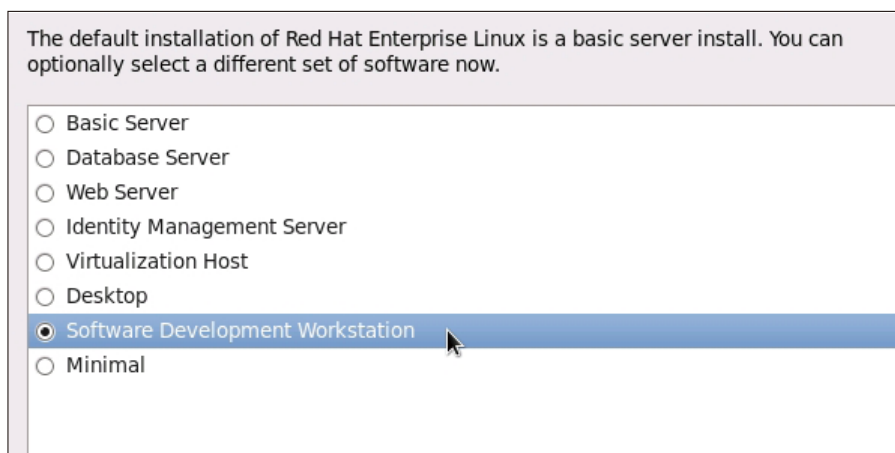


图 3-3 在 RHEL 6.3 安装中选择服务器类型

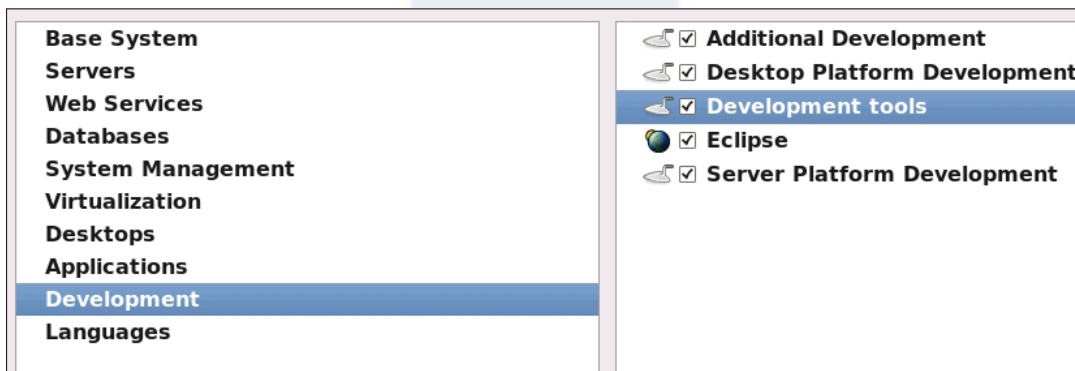


图 3-4 Software Development Workstation 默认选择的具体组件

然后，继续进行后面安装流程，可以安装相应的软件包，安装过程的一个快照如图 3-5 所示。



图 3-5 RHEL 6.3 安装过程快照

在安装完所有软件包后，系统会提示安装完成需要重启系统，重启后即可进入到 RHEL 6.3 系统中。至此，Linux 系统就安装完毕了，这就是在本书中作为宿主机（Host）的操作系统，后面的编译和实验都是在这个宿主机上进行的（当然，我们会使用本章讲述的自己编译的 kernel 和 qemu-kvm 来进行实验）。

3.3 编译和安装 KVM

3.3.1 下载 KVM 源代码

KVM 作为 Linux kernel 中的一个 module 而存在，是从 Linux 2.6.20 版本开始被完全正式加入到内核的主干开发和正式发布代码中。所以，只需要下载 2.6.20 版本之后 Linux kernel 代码即可编译和使用 KVM。当然，如果是学习 KVM，推荐使用最新正式发布或者正在开发中的 kernel 版本，如果是实际部署到生产环境中，还需要自己选择适合的稳定版本进行详尽的功能和性能测试。如果你想使用最新的处于开发中的 KVM 代码，你需要自己下载 KVM 的代码仓库，本节就是以此为例来讲解的。

总的来说，下载最新 KVM 源代码，主要有如下三种方式：

- ❑ 下载 KVM 项目开发中的代码仓库 `kvm.git`。
- ❑ 下载 Linux 内核的代码仓库 `linux.git`。
- ❑ 打包下载 Linux 内核的源代码（Tarball^[3] 格式）。

1. 下载 `kvm.git`

KVM 项目的代码是托管在 Linux 内核官方源码网站 <http://git.kernel.org> 上的，可以到上面去查看和下载。该网页上 `virt/kvm/kvm.git` 即是 KVM 项目的代码，它是最新的功能最丰富的 KVM 源代码库（尽管并非最稳定的）。目前，`kvm.git` 的最主要维护者（maintainer）是来自 Redhat 公司的 Gleb Natapov 和 PaoloBonzini。从 <http://git.kernel.org/?p=virt/kvm/kvm.git> 网页可以看到，`kvm.git` 下载链接有如下 3 个 URL，可用于下载最新的 KVM 的开发代码仓库。

```
git://git.kernel.org/pub/scm/virt/kvm/kvm.git
http://git.kernel.org/pub/scm/virt/kvm/kvm.git
https://git.kernel.org/pub/scm/virt/kvm/kvm.git
```

从这 3 个 URL 下载的内容完全一致，根据自己实际情况选择其中一个下载即可。Linux 内核相关的项目一般都使用 Git^[4] 作为源代码管理工具，KVM 当然也是用 Git 管理源码的。可以使用 `git clone` 命令来下载 KVM 的源代码，也可以使用 Git 工具的其他命令对源码进行各种管理，这里不详述 Git 的各种命令。

`kvm.git` 的下载方式和过程，为如下命令行所示：

```
[root@jay-linux ~] cd kvm_demo
[root@jay-linux kvm_demo]# git clone \
git://git.kernel.org/pub/scm/virt/kvm/kvm.git kvm.git
```

```

Initialized empty Git repository in /root/kvm_demo/kvm.git/.git/
remote: Counting objects: 2556717, done.
remote: Compressing objects: 100% (399423/399423), done.
Receiving objects: 100% (2556717/2556717), 517.22 MiB | 11.21 MiB/s, done.
remote: Total 2556717 (delta 2141777), reused 2546109 (delta 2131175)
Resolving deltas: 100% (2141777/2141777), done.
[root@jay-linux kvm_demo]# cd kvm.git
[root@jay-linux kvm.git]# pwd
/root/kvm_demo/kvm.git

```

2. 下载 linux.git

Linux 内核的官方网站为 <http://kernel.org>，其中源代码管理网为 <http://git.kernel.org>，可以在此网站上找到最新的 linux.git 代码。在源码管理网站上，我们看到有多个 linux.git，我们选择 Linus Torvalds^[5] 的源码库（也即是 Linux 内核的主干）。在内核源码的网页 <http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git> 中可以看到，其源码仓库也有如下 3 个链接可用：

```

git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
http://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git

```

这 3 个 URL 中源码内容是完全相同的，可以用 `git clone` 命令复制到本地，其具体操作方式与前一种（kvm.git）的下载方式完全一样。

3. 下载 Linux 的 Tarball

在 Linux 官方网站（<http://kernel.org/>）上，也提供 Linux 内核的 Tarball 文件下载；除了在其首页上单击一些 Tarball 之外，也可以到如下网址下载 Linux 内核的各个版本的 Tarball。

❑ <ftp://ftp.kernel.org/pub/linux/kernel/>

❑ <http://www.kernel.org/pub/linux/kernel/>

kernel.org 还提供一种 rsync 的方式下载，此处不详细叙述，请参见其官网首页的提示。

以用 `wget` 下载 linux-3.4.1.tar.gz 为例，有如下的命令行代码：

```

[root@jay-linux kvm_demo]# wget\
http://www.kernel.org/pub/linux/kernel/v3.x/linux-3.4.1.tar.gz
--2012-06-05 23:06:26--
http://www.kernel.org/pub/linux/kernel/v3.x/linux-3.4.1.tar.gz
Resolving www.kernel.org... 149.20.20.133, 149.20.4.69
Connecting to www.kernel.org|149.20.20.133|:80...connected
Length: 101055487 (96M) [application/x-gzip]
Saving to: "linux-3.4.1.tar.gz"

100%[=====] 101,055,487 9.13M/s in 11s

2012-06-05 23:06:37 (8.79 MB/s) - "linux-3.4.1.tar.gz" saved
[101055487/101055487]
[root@jay-linux kvm_demo]# ls -l
total 98692

```

```
drwxr-xr-x 24 root root      4096 Jun  5 23:05 kvm.git
-rw-r--r--  1 root root 101055487 Jun  5 02:36 linux-3.4.1.tar.gz
[root@jay-linux kvm_demo]# tar -zxvf linux-3.4.1.tar.gz
[root@jay-linux kvm_demo]# ls -l
total 98696
drwxr-xr-x 24 root root      4096 Jun  5 23:05 kvm.git
drwxrwxr-x 23 root root      4096 Jun  1 15:18 linux-3.4.1
-rw-r--r--  1 root root 101055487 Jun  5 02:36 linux-3.4.1.tar.gz
```

4. 通过 kernel.org 的镜像站点下载

由于 Linux 的源代码量比较大，如果只有美国一个站点可供下载，那么速度会较慢，服务器压力也较大。所以，kernel.org 在世界上多个国家和地区都有一些镜像站点，而且一些 Linux 开源社区的爱好者们也自发建立了不少 kernel.org 的镜像，在中国的镜像站点中，推荐大家从如下两个镜像站点下载 Linux 相关的代码及其他源码（在本书写作之时，它们并没有提供 git 形式的代码仓库 kvm.git 或 linux.git 的下载），访问速度比较快。

❑ 一个是清华大学开源镜像站：<http://mirror.tuna.tsinghua.edu.cn/>，其中的链接地址 <http://mirror.tuna.tsinghua.edu.cn/kernel/linux/kernel/> 与 <http://www.kernel.org/pub/linux/kernel/> 就是同步的，用起来比较方便。

❑ 另外一个是一个北京交通大学的一个开源镜像站：

<http://mirror.bjtu.edu.cn/kernel/linux/kernel/>

还有如下两个镜像站可以推荐给大家：

❑ 网易开源镜像站，<http://mirrors.163.com/>

❑ 搜狐开源镜像站，<http://mirrors.sohu.com/>

（在本书写作之时，网易和搜狐的两个镜像站主要提供了一些 Linux 发行版的 ISO 及其软件仓库，而没有提供 linux kernel 的源码下载。）

3.3.2 配置 KVM

上面三种方式下载的源代码都是可以同样地进行配置和编译，本章以开发中的最新源代码仓库 kvm.git 来讲解 KVM 的配置和编译等。KVM 是作为 Linux 内核中的一个 module 存在的，而 kvm.git 是一个包含了最新的 KVM 模块开发中代码完整的 Linux 内核源码仓库。它的配置方式，与普通的 Linux 内核配置完全一样，只是需要注意将 KVM 相关的配置选择为编译进内核或者编译为模块。

在 kvm.git（Linux kernel）代码目录下，运行“make help”命令可以得到一些关于如何配置和编译 kernel 的帮助手册，命令行如下：

```
[root@jay-linux kvm.git]# make help
Cleaning targets:
  clean          - Remove most generated files but keep the config and
                  enough build support to build external modules
  mrproper       - Remove all generated files + config + various backup files
```

```

distclean      - mrproper + remove editor backup and patch files
Configuration targets:
config         - Update current config utilising a line-oriented program
nconfig        - Update current config utilising a ncurses menu based program
menuconfig     - Update current config utilising a menu based program
xconfig        - Update current config utilising a QT based front-end

<!-- 此处省略数十行帮助信息 -->

make W=n      [targets] Enable extra gcc checks, n=1,2,3 where
                1: warnings which may be relevant and do not occur too often
                2: warnings which occur quite often but may still be relevant
                3: more obscure warnings, can most likely be ignored
                Multiple levels can be combined with W=12 or W=123

Execute "make" or "make all" to build all targets marked with [*]
For further info see the ./README file

```

对 KVM 或 Linux 内核配置时常用的配置命令的一些解释如下：

(1) make config

基于文本的最为传统的也是最为枯燥的一种配置方式，但是它可以适用于任何情况之下，这种方式会为每一个内核支持的特性向用户提问，如果用户回答“y”，则把特性编译进内核；回答“m”，则该特性作为模块进行编译；回答“n”，则表示不对该特性提供支持。输入“？”则显示该特性的帮助信息，在了解之后再决定处理该特性的方式；在回答每个问题前，必须考虑清楚，如果在配置过程中因为失误而给了错误的回答，就只能按“ctrl+c”强行退出然后重新配置了。

(2) make oldconfig

make oldconfig 和 make config 类似，但是它的作用是在现有的内核设置文件基础上建立一个新的设置文件，只会向用户提供有关新内核特性的问题，在新内核升级的过程中，make oldconfig 非常有用，用户将现有的配置文件 .config 复制到新内核的源码中，执行 make oldconfig，此时，用户只需要回答那些针对新增特性的问题。

(3) make silentoldconfig

和上面 make oldconfig 一样，但在屏幕上不再出现已在 .config 中配置好的选项。

(4) make menuconfig

基于终端的一种配置方式，提供了文本模式的图形用户界面，用户可以通过光标移动来浏览所支持的各种特性。使用这用配置方式时，系统中必须安装有 ncurses 库，否则会显示“Unable to find the ncurses libraries”的错误提示。其中“y”、“n”、“m”、“？”键的选择功能与前面 make config 中介绍的一致。

(5) make xconfig

基于 XWindow 的一种配置方式，提供了漂亮的配置窗口，不过只有能够在 X Server 上运行 X 桌面应用程序时才能够使用，它依赖于 QT，如果系统中没有安装 QT 库，则会出现“Unable to find any QT installation”的错误提示。

(6) make gconfig

与 make xconfig 类似，不同的是 make gconfig 依赖于 GTK 库。

(7) make defconfig

按照内核代码中提供的默认配置文件对内核进行配置（在 Intel x86_64 平台上，默认配置为 arch/x86/configs/x86_64_defconfig），生成 .config 文件可以用作初始化配置，然后再使用 make menuconfig 进行定制化配置。

(8) make allyesconfig

尽可能多地使用“y”设置内核选项值，生成的配置中包含了全部的内核特性。

(9) make allnoconfig

除必须的选项外，其他选项一律不选（常用于嵌入式 Linux 系统的编译）。

(10) make allmodconfig

尽可能多地使用“m”设置内核选项值来生成配置文件。

(11) make localmodconfig

会执行 lsmod 命令查看当前系统中加载了哪些模块 (Modules)，并将原来的 .config 中不需要的模块去掉，仅保留前面 lsmod 命令查出来的这些模块，从而简化了内核的配置过程。这样做确实方便了很多，但是也有个缺点：该方法仅能使编译出的内核支持当前内核已经加载的模块。因为该方法使用的是 lsmod 查询得到的结果，如果有的模块当前没有加载，那么就不会编到新的内核中。

下面以 make menuconfig 为例介绍一下如何选择 KVM 相关的配置。运行 make menuconfig 后显示的界面如图 3-6 所示。

```

qoooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
  General setup --->
  [*] Enable loadable module support --->
  *- Enable the block layer --->
    Processor type and features --->
    Power management and ACPI options --->
    Bus options (PCI etc.) --->
    Executable file formats / Emulations --->
  [*] Networking support --->
    Device Drivers --->
    Firmware Drivers --->
    File systems --->
    Kernel hacking --->
    Security options --->
  *- Cryptographic API --->
  [*] Virtualization --->
    Library routines --->
  ---
    Load an Alternate Configuration File
    Save an Alternate Configuration File

qoooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
qoooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
                                     <Select>  < Exit >  < Help >

```

图 3-6 make menuconfig 命令的选择界面

选择了 Virtualization 之后，进入其中进行详细配置，包括选中 KVM、选中对处理器的支持（比如：KVM for Intel processors support，KVM for AMD processors support）等，如图 3-7 所示。

在配置完成之后，就会在 kvm.git 的目录下面生成一个 .config 文件，最好检查一下 KVM 相关的配置是否正确。在本次配置中，与 KVM 直接相关的几个配置项的主要情况如下：

```
CONFIG_HAVE_KVM=y
CONFIG_HAVE_KVM_IRQCHIP=y
CONFIG_HAVE_KVM_EVENTFD=y
CONFIG_KVM_APIC_ARCHITECTURE=y
CONFIG_KVM_MMIO=y
CONFIG_KVM_ASYNC_PF=y
CONFIG_HAVE_KVM_MSI=y
CONFIG_VIRTUALIZATION=y
CONFIG_KVM=m
CONFIG_KVM_INTEL=m
# CONFIG_KVM_AMD is not set
CONFIG_KVM_MMU_AUDIT=y
```

```
[*] built-in [ ] excluded <M> module < > module capable

--- Virtualization
<M> Kernel-based Virtual Machine (KVM) support
<M>   KVM for Intel processors support
< >   KVM for AMD processors support
[*]   Audit KVM MMU
<M>   Host kernel accelerator for virtio net (EXPERIMENTAL)
```

图 3-7 Virtualization 中的配置选项

3.3.3 编译 KVM

在对 KVM 源代码进行了配置之后，编译 KVM 就是一件比较容易的事情了。它的编译过程就完全是一个普通 Linux 内核编译的过程，需要经过编译 kernel、编译 bzImage 和编译 module 等三个步骤。编译 bzImage 这一步不是必须的，在本章示例中，config 中使用了 initramfs，所以这里需要这个 bzImage 用于生成 initramfs image。另外，在最新的 Linux kernel 代码中，根据 Makefile 中的定义可以看出，直接执行“make”或“make all”命令就可以将这里提及的 3 个步骤全部包括在内。本节是为了更好地展示编译的过程，才将编译的步骤分为这三步来解释。

第一步，编译 kernel 的命令为“make vmlinux”，其编译命令和输出如下：

```
[root@jay-linux kvm.git]# makevmlinux -j 20
make[1]: Nothing to be done for 'all'.
HOSTCC arch/x86/tools/relocs
CHK include/linux/version.h
HOSTCC scripts/basic/fixdep
```

```

CHK      include/generated/utsrelease.h
HOSTCC   scripts/kallsyms
HOSTCC   scripts/pnmtologo
HOSTCC   scripts/genksyms/genksyms.o
CC       scripts/mod/empty.o

<!-- 此处省略数千行编译时的输出信息 -->

LINK     vmlinux
LD       vmlinux.o
MODPOST  vmlinux.o
GEN      .version
CHK      include/generated/compile.h
UPD      include/generated/compile.h
CC       init/version.o
LD       init/built-in.o
KSYM     .tmp_kallsyms1.o
KSYM     .tmp_kallsyms2.o

LD       vmlinux # 这里就是编译、链接后生成了启动所需的 Linux kernel 文件

SORTEX   vmlinux
sort done marker at 9725f0
SYSMAP   System.map

```

其中，编译命令中“-j”参数并非必须的，它是让 make 工具用多进程来编译，比如上面命令中提到的“-j 20”，会让 make 工具最多创建 20 个 GCC 进程同时来执行编译任务。在一个比较空闲的系统上面，有一个推荐值作为 -j 参数的值，即大约为 2 倍于系统上的 CPU 的 core 的数量（CPU 超线程也算 core）。

第二步，执行编译 bzImage 的命令“make bzImage”，其输出如下：

```

[root@jay-linux kvm.git]# make bzImage
make[1]: Nothing to be done for 'all'.
make[1]: Nothing to be done for 'relocs'.
CHK      include/linux/version.h
CHK      include/generated/utsrelease.h
CALL     scripts/checksyscalls.sh
CHK      include/generated/compile.h
make[3]: 'arch/x86/realmode/rm/realmode.bin' is up to date.
CC       arch/x86/boot/a20.o
AS       arch/x86/boot/bioscall.o

<!-- 此处省略数十行编译时的输出信息 -->

CC       arch/x86/boot/video-bios.o
LD       arch/x86/boot/setup.elf
OBJCOPY  arch/x86/boot/setup.bin
OBJCOPY  arch/x86/boot/vmlinux.bin
HOSTCC   arch/x86/boot/tools/build
BUILD    arch/x86/boot/bzImage # 这里生成了我们需要的 bzImage 文件
Kernel: arch/x86/boot/bzImage is ready  (#4)

```

第三步，编译 kernel 和 bzImage 之后编译内核的模块，命令为“make modules”，其命令行输出如下：

```
[root@jay-linux kvm.git]# make modules -j 20
make[1]: Nothing to be done for 'all'.
make[1]: Nothing to be done for 'relocs'.
CHK      include/linux/version.h
CHK      include/generated/utsrelease.h
CALL     scripts/checksyscalls.sh
CC [M]   arch/x86/kvm/vmx.o
```

<!-- 此处省略数千行编译时的输出信息 -->

```
IHEX     firmware/mts_gsm.fw
IHEX     firmware/mts_edge.fw
H16TOFW  firmware/edgeport/boot.fw
H16TOFW  firmware/edgeport/boot2.fw
H16TOFW  firmware/edgeport/down.fw
H16TOFW  firmware/edgeport/down2.fw
IHEX     firmware/edgeport/down3.bin
IHEX2FW  firmware/whiteheat_loader.fw
IHEX2FW  firmware/whiteheat.fw
IHEX2FW  firmware/keyspan_pda/keyspan_pda.fw
IHEX2FW  firmware/keyspan_pda/xircom_pgs.fw
```

3.3.4 安装 KVM

编译完 KVM 之后，下面介绍如何安装 KVM。

KVM 的安装包括两个步骤：module 的安装、kernel 与 initramfs 的安装。

(1) 安装 module

通过“make modules_install”命令可以将编译好的 module 安装到相应的目录之中，在默认情况下 module 被安装到 /lib/modules/\$kernel_version/kernel 目录之中。

```
[root@jay-linux kvm.git]# make modules_install
INSTALL arch/x86/crypto/aes-x86_64.ko
INSTALL arch/x86/kernel/microcode.ko
INSTALL arch/x86/kvm/kvm-intel.ko
INSTALL arch/x86/kvm/kvm.ko
INSTALL arch/x86/oprofile/oprofile.ko
```

<!-- 此处省略千余行安装时的输出信息 -->

```
INSTALL /lib/firmware/edgeport/down3.bin
INSTALL /lib/firmware/whiteheat_loader.fw
INSTALL /lib/firmware/whiteheat.fw
INSTALL /lib/firmware/keyspan_pda/keyspan_pda.fw
INSTALL /lib/firmware/keyspan_pda/xircom_pgs.fw
DEPMOD 3.5.0-rc2+
```

安装好 module 之后，可以查看一下相应的安装路径，可看到 kvm 模块也已经安装，如下所示：

```
[root@jay-linux kvm.git]# ls -l /lib/modules/3.5.0-rc2+/kernel/
total 28
drwxr-xr-x  3 root root 4096 Jun 13 08:58 arch
drwxr-xr-x  3 root root 4096 Jun 13 08:58 crypto
drwxr-xr-x 36 root root 4096 Jun 13 08:58 drivers
drwxr-xr-x 25 root root 4096 Jun 13 08:58 fs
drwxr-xr-x  5 root root 4096 Jun 13 08:58 lib
drwxr-xr-x 22 root root 4096 Jun 13 08:58 net
drwxr-xr-x  9 root root 4096 Jun 13 08:58 sound
[root@jay-linux kvm.git]# ls -l
/lib/modules/3.5.0-rc2+/kernel/arch/x86/kvm/total 528
-rw-r--r-- 1 root root 176861 Jun 13 08:58 kvm-intel.ko
-rw-r--r-- 1 root root 357867 Jun 13 08:58 kvm.ko
```

(2) 安装 kernel 和 initramfs

通过“make install”命令可以安装 kernel 和 initramfs，命令行输出如下：

```
[root@jay-linux kvm.git]# make install
sh /root/kvm_demo/kvm.git/arch/x86/boot/install.sh 3.5.0-rc2+
arch/x86/boot/bzImage \
    System.map "/boot"
[root@jay-linux kvm.git]# ls -l /boot/
-rw-r--r-- 1 root root 2775664 Jun 13 08:45 vmlinuz-3.5.0-rc2+
-rw-r--r-- 1 root root 1825462 Jun 13 08:45 System.map-3.5.0-rc2+
lrwxrwxrwx 1 root root      24 Jun 13 08:45 vmlinuz ->
/boot/vmlinuz-3.5.0-rc2+
lrwxrwxrwx 1 root root      27 Jun 13 08:45 System.map ->
/boot/System.map-3.5.0-rc2+
dr-xr-xr-x 4 root root 4096 Jun 13 08:45 ./
-rw-r--r-- 1 root root 4672632 Jun 13 08:45 initramfs-3.5.0-rc2+.img
drwxr-xr-x 2 root root 4096 Jun 13 08:45 grub/
```

可见，在 /boot 目录下生成了内核（vmlinuz）和 initramfs 等内核启动所需的文件。

在运行 make install 之后，grub 配置文件（如：/boot/grub/grub.conf）中也自动添加了一个 grub 选项，如下所示：

```
title Red Hat Enterprise Linux Server (3.5.0-rc2+)
    root (hd0,0)
    kernel /boot/vmlinuz-3.5.0-r2+ ro
root=UUID=1a65b4bb-cd9b-4bbf-97ff-7e1f7698d3db r
initrd /boot/initramfs-3.5.0-r2+.img
```

检查了 grub 之后，重新启动系统，选择刚才为了 KVM 而编译、安装的内核启动。

系统启动后，登录进入系统，在通常情况下，系统启动时默认已经加载了 kvm 和 kvm_intel 这两个模块；如果没有加载，请手动用 modprobe 命令依次加载 kvm 和 kvm_intel 模块。

```
[root@jay-linux ~]# modprobe kvm
[root@jay-linux ~]# modprobe kvm_intel
[root@jay-linux ~]# lsmod | grep kvm
kvm_intel          112487  0
kvm                206544  1 kvm_intel
```

确认 KVM 相关的模块加载成功后，检查 /dev/kvm 这个文件，它是 kvm 内核模块提供给用户空间的 qemu-kvm 程序使用的一个控制接口，它提供了客户机（Guest）操作系统运行所需要的模拟和实际的硬件设备环境。

```
[root@jay-linux ~]# ls -l /dev/kvm
crw-rw---- 1 root root 10, 232 Jun  6 15:45 /dev/kvm
```

3.4 编译和安装 qemu-kvm

除了在内核空间的 kvm 模块之外，在用户空间需要 QEMU^[6] 来模拟所需要 CPU 和设备模型以及用于启动客户机进程，这样才有了一个完整的 KVM 运行环境。而 qemu-kvm 是为了针对 KVM 专门做了修改和优化的 QEMU 分支^[7]，在本书写作的 2012 年，qemu-kvm 分支里面的小部分特性还没有完全合并进入到 qemu 的主干代码之中，所以本书中采用 qemu-kvm 来讲解。

在编译和安装了 KVM 并且启动到编译的内核之后，下面来看一下 qemu-kvm 的编译和安装。

3.4.1 下载 qemu-kvm 源代码

目前的 QEMU 项目针对 KVM 的代码分支 qemu-kvm 也是由 Redhat 公司的 Gleb Natapov 和 Paolo Bonzini 作维护者（Maintainer），代码也是托管在 kernel.org 上。

qemu-kvm 开发代码仓库的网页连接为：<http://git.kernel.org/?p=virt/kvm/qemu-kvm.git>
其中，可以看到有如下 3 个 URL 连接可供下载开发中的最新 qemu-kvm 的代码仓库。

```
git://git.kernel.org/pub/scm/virt/kvm/qemu-kvm.git
http://git.kernel.org/pub/scm/virt/kvm/qemu-kvm.git
https://git.kernel.org/pub/scm/virt/kvm/qemu-kvm.git
```

可以根据自己实际需要选择 3 个中任一个用 git clone 命令下载即可，它们是完全一样的。

另外，可以到 sourceforge.net 的如下链接中根据需要下载 qemu-kvm 各个发布版本的代码压缩包（笔者建议使用最新的正式发布版本，因为它的功能更多，同时也比较稳定）：

<http://sourceforge.net/projects/kvm/files/qemu-kvm/>

<http://qemu-project.org/Download>（从 2013 年开始直接到这里下载普通的 QEMU）

在本节后面讲解编译时，是以下载开发中的最新的 qemu-kvm.git 为例的，获取其代码仓库过程如下：

```
[root@jay-linux kvm_demo]# git clone\
git://git.kernel.org/pub/scm/virt/kvm/qemu-kvm.git qemu-kvm.git
Initialized empty Git repository in /root/kvm_demo/qemu-kvm.git/.git/
remote: Counting objects: 145222, done.
remote: Compressing objects: 100% (35825/35825), done.
remote: Total 145222 (delta 114656), reused 137663 (delta 107444)
Receiving objects: 100% (145222/145222), 40.83 MiB | 10.33 MiB/s, done.
```



```
Resolving deltas: 100% (114656/114656), done.
[root@jay-linux kvm_demo]# cdqemu-kvm.git
[root@jay-linux kvm.git]# pwd
/root/kvm_demo/qemu-kvm.git
```

3.4.2 配置和编译 qemu-kvm

qemu-kvm 的配置并不复杂，在通常情况下，可以直接运行代码仓库中 `configure` 文件进行配置即可。当然，如果对其配置并不熟悉，可以运行 “`./configure --help`” 命令查看配置的一些选项及其帮助信息。

显示配置的帮助手册信息如下：

```
[root@jay-linux qemu-kvm.git]# ./configure --help
Usage: configure [options]
Options: [defaults in brackets after descriptions]

Standard options:
  --help                print this message
  --prefix=PREFIX       install in PREFIX [/usr/local]
  --interp-prefix=PREFIX where to find shared libraries, etc.
                        use %M for cpu name [/usr/gnemul/qemu-%M]
  --target-list=LIST    set target list (default: build everything)
                        Available targets: i386-softmmu x86_64-softmmu
<!-- 此处省略百余行帮助信息的输出 -->

  --disable-guest-agent  disable building of the QEMU Guest Agent
  --enable-guest-agent  enable building of the QEMU Guest Agent
  --with-coroutine=BACKEND coroutine backend. Supported options:
                        gthread, ucontext, sigaltstack, windows
```

NOTE: The object files are built at the place where configure is launched

执行 `configure` 文件进行配置的过程如下：

```
[root@jay-linux qemu-kvm.git]# ./configure
Install prefix      /usr/local
BIOS directory      /usr/local/share/qemu
binary directory    /usr/local/bin
library directory   /usr/local/lib
include directory    /usr/local/include
config directory    /usr/local/etc
Manual directory     /usr/local/share/man
ELF interp prefix    /usr/gnemul/qemu-%M
Source path         /root/kvm_demo/qemu-kvm.git
C compiler           gcc
Host C compiler      gcc

<!-- 此处省略数十行 -->

VNC support         yes      # 通常需要通过 VNC 连接到客户机中
```

```
<!-- 此处省略十余行 -->
KVM support          yes          # 这是对 KVM 的支持
TCG interpreter      no
KVM device assign. yes          # 这是对 KVM 中 VT-d 功能的支持
<!-- 此处省略十余行 -->
OpenGL support      yes
libiscsi support    no
build guest agent   yes
coroutine backend   ucontext
```

需要注意的是，上面命令行输出的 KVM 相关的选项需要是配置为 yes，另外，一般 VNC 的支持也是配置为 yes 的（因为通常需要用 VNC 连接到客户机中）。

经过配置之后，进行编译就很简单了，直接执行 make 即可进行编译，如下所示：

```
[root@jay-linux qemu-kvm.git]# make -j 20
GEN    config-host.h
GEN    trace.h
GEN    qemu-options.def
GEN    qmp-commands.h
GEN    qapi-types.h
GEN    qapi-visit.h
GEN    tests/test-qapi-types.h
GEN    tests/test-qapi-visit.h
GEN    tests/test-qmp-commands.h
CC     libcacard/cac.o
CC     libcacard/event.o
<!-- 此处省略数百行的编译时输出信息 -->
CC     x86_64-softmmu/target-i386/cpu.o
CC     x86_64-softmmu/target-i386/machine.o
CC     x86_64-softmmu/target-i386/arch_memory_mapping.o
CC     x86_64-softmmu/target-i386/arch_dump.o
CC     x86_64-softmmu/target-i386/kvm.o
CC     x86_64-softmmu/target-i386/hyperv.o
LINK   x86_64-softmmu/qemu-system-x86_64
```

可以看到，最后有编译生成 qemu-system-x86_64 文件，它就是我们常用的 qemu-kvm 的命令行工具（在多数 Linux 发行版中自带的 qemu-kvm 软件包的命令行是 qemu-kvm，只是名字不同而已）。

3.4.3 安装 qemu-kvm

编译完成之后，运行“make install”命令即可安装 qemu-kvm，其过程如下：

```
[root@jay-linux qemu-kvm.git]# make install | tee make-install.log
install -d -m 0755 "/usr/local/share/qemu"
install -d -m 0755 "/usr/local/etc/qemu"
install -c -m 0644
/root/kvm_demo/qemu-kvm.git/sysconfigs/target/target-x86_64.conf
```

```

"/usr/local/etc/qemu"
install -c -m 0644
/root/kvm_demo/qemu-kvm.git/sysconfigs/target/cpus-x86_64.conf
"/usr/local/share/qemu"
install -d -m 0755 "/usr/local/bin"
install -c -m 0755 vscclient qemu-ga qemu-nbd qemu-img qemu-io "/usr/local/bin"
install -d -m 0755 "/usr/local/libexec"

<!-- 此处省略数行的安装时输出信息 -->

make[1]: Entering directory
'/root/kvm_demo/qemu-kvm.git/x86_64-softmmu'
install -m 755 qemu-system-x86_64 "/usr/local/bin"
strip "/usr/local/bin/qemu-system-x86_64"
make[1]: Leaving directory
'/root/kvm_demo/qemu-kvm.git/x86_64-softmmu'

```

qemu-kvm 安装过程的主要任务有这么几个：创建 qemu 的一些目录，复制一些配置文件到相应的目录下，复制一些 firmware 文件（如：sgabios.bin, kvmvapic.bin）到目录下以便 qemu-kvm 的命令行启动时可以找到对应的固件提供给客户机使用，复制 keymaps 到相应的目录下以便在客户机中支持各种所需键盘类型，复制 qemu-system-x86_64、qemu-img 等可执行程序到对应的目录下。下面的一些命令行检查了 qemu-kvm 被安装了之后的系统状态。

```

[root@jay-linux qemu-kvm.git]# which qemu-system-x86_64
/usr/local/bin/qemu-system-x86_64
[root@jay-linux qemu-kvm.git]# which qemu-img
/usr/local/bin/qemu-img
[root@jay-linux qemu-kvm.git]# ls /usr/local/share/qemu/
bamboo.dtb      mpc8544ds.dtb      petalogix-ml605.dtb
pxe-pcnet.rom   slof.bin            vgabios-vmware.bin
bios.bin        multiboot.bin       petalogix-s3adsp1800.dtb
pxe-rtl8139.rom spapr-rtas.bin
cpus-x86_64.conf openbios-ppc         ppc_rom.bin
pxe-virtio.rom  vgabios.bin
keymaps        openbios-sparc32    pxe-e1000.rom
qemu-icon.bmp   vgabios-cirrus.bin
kvmvapic.bin    openbios-sparc64    pxe-eeepro100.rom
s390-zipl.rom   vgabios-qxl.bin
linuxboot.bin   palcode-clipper     pxe-ne2k_pci.rom
sgabios.bin     vgabios-stdvga.bin
[root@jay-linux qemu-kvm.git]# ls /usr/local/share/qemu/keymaps/
ar      common  de      en-gb  es  fi  fr      fr-ca  hr  is  ja  lv
modifiers nl-be  pl      pt-br  sl  th
bepo    da      de-ch  en-us  et  fo  fr-be  fr-ch  hu  it  lt  mk  nl
no      pt  ru      sv  tr

```

由于 qemu-kvm 是用户空间的程序，安装之后不用重启系统，直接用 qemu-system-x86_64、qemu-img 这样的命令行工具即可使用 qemu-kvm 了。

3.5 安装客户机

安装客户机（Guest）之前，我们需要创建一个镜像文件或者磁盘分区等来存储客户机中

的系统和文件，关于客户机镜像有很多制作和存储方式（将在后面的第4章中进行详细的介绍），本节只是为了快速地演示安装一个客户机，采用了本地创建一个镜像文件，然后让镜像文件作为客户机的硬盘，将客户机操作系统（以 RHEL6.3 为例）安装在其中。

首先，需要创建一个镜像文件，可以使用 dd 工具，如下的命令行创建了一个 8GB 大小的镜像文件 rhel6u3.img：

```
[root@jay-linux kvm_demo]# dd if=/dev/zero of=rhel6u3.img bs=1M\ count=8192
8192+0 records in
8192+0 records out
8589934592 bytes (8.6 GB) copied, 76.9331 s, 112 MB/s
```

然后，准备一个 RHEL6.3 安装所需的 ISO 文件，如下所示：

```
[root@jay-linux kvm_demo]# ls RHEL6.3-Server-x86_64-DVD1.iso
RHEL6.3-Server-x86_64-DVD1.iso
```

启动客户机，并在其中用准备好的 ISO 安装系统，命令行如下：

```
[root@jay-linux kvm_demo]# qemu-system-x86_64 -m 2048 -smp 4 -boot
order=cd -hda /root/kvm_demo/rhel6u3.img -cdrom /root/kvm_demo/
RHEL6.3-Server-x86_64-DVD1.iso
VNC server running on ':::1:5900'
```

其中，-m 2048 是给客户机分配 2048MB 内存，-smp 4 是给客户机分配 4 个 CPU，-boot order=cd 是指定系统的启动顺序为光驱（c: CD-ROM）、硬盘（d: hard Disk），-hda ** 是分配给客户机的 IDE 硬盘（即前面准备的镜像文件），-cdrom ** 是分配客户机的光驱。在默认情况下，QEMU 会启动一个 VNC server 端口（如上面的 :::1:5900），可以用 vncviewer^[8] 工具来连接到 QEMU 的 VNC 端口查看客户机。

通过启动时的提示，这里可以使用“vncviewer :5900”命令连接到 QEMU 启动的窗口。根据命令行制定的启动顺序，当有 CDROM 时，客户机默认会从光驱引导，启动后即可进入到客户机系统安装界面如图 3-8 所示。

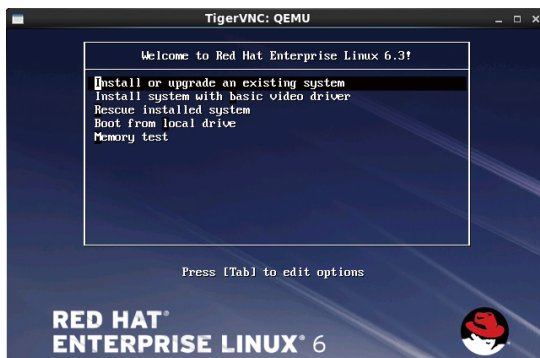


图 3-8 客户机安装的选择界面

可以选择 Install 安装客户机操作系统，和安装普通 Linux 系统类似，根据需要做磁盘分

区、选择需要的软件包等，安装过程中的一个快照如图 3-9 所示。

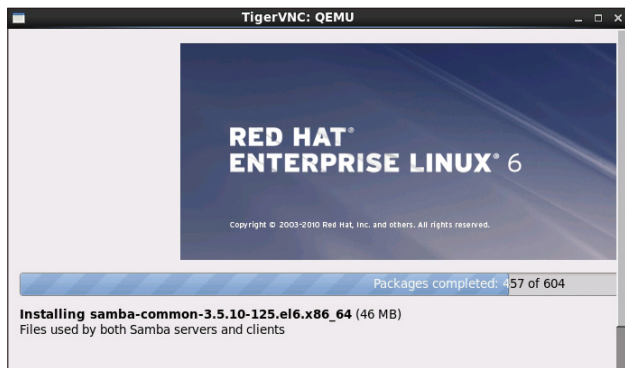


图 3-9 客户机安装过程的快照

在系统安装完成后，客户机中安装程序提示，如图 3-10 所示。

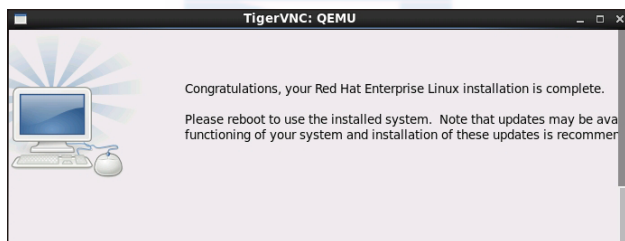


图 3-10 客户机安装完成后的提示信息

和安装普通的 Linux 系统一样，完成安装后，重启系统即可进入到刚才安装的客户机操作系统。

3.6 启动第一个 KVM 客户机

在安装好了系统之后，就可以使用镜像文件来启动并登录到自己安装的系统之中了。通过如下的命令行即可启动一个 KVM 的客户机。

```
[root@jay-linux kvm_demo]# qemu-system-x86_64 -m 2048 -smp 4 -hda
/root/kvm_demo/rhel6u3.img
VNC server running on ':::1:5900'
```

用 vncviewer 命令（此处命令为 vncviewer :5900）查看客户机的启动情况，客户机启动过程中的状态，如图 3-11 所示。

客户机启动完成后的登录界面如图 3-12 所示。

在通过 VNC 链接到 QEMU 窗口后，可以按快捷键“Ctrl+Alt+2”切换到 QEMU 监视器窗口，在监视器窗口中可以执行一些命令，比如执行“info kvm”命令来查看当前 QEMU 是否使用着 KVM，如图 3-13 所示（显示为 kvm support: enabled）。

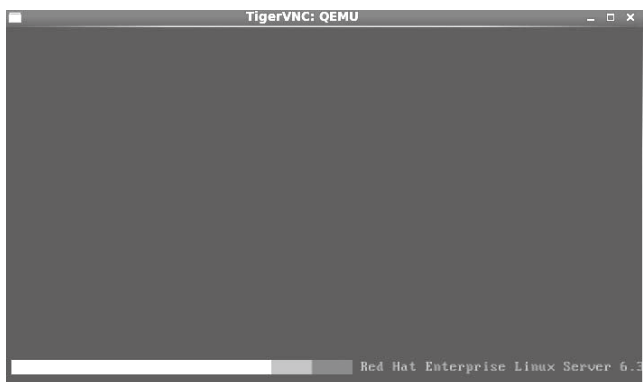


图 3-11 客户机启动中状态

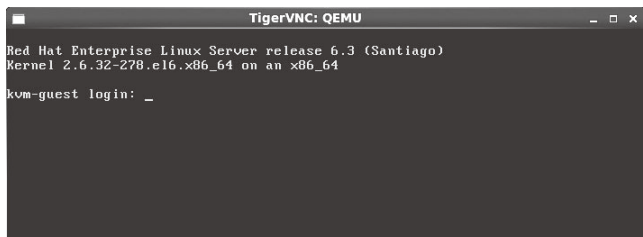


图 3-12 客户机启动后的登录界面

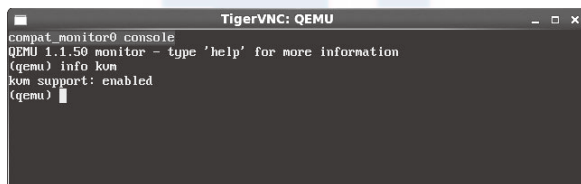


图 3-13 QEMU Monitor 中 “info kvm” 命令

用快捷键“Ctrl+Alt+1”切换回普通的客户机查看窗口，就可以登录或继续使用客户机系统了。至此，你就已经启动了属于自己的第一个 KVM 客户机，尽情享受 KVM 虚拟化带来的快乐吧！

3.7 本章小结

本章主要讲解了如何一步一步地构建属于自己的 KVM 虚拟化环境。硬件和 BIOS 中虚拟化技术的支持是 KVM 运行的先决条件，编译 kvm 和 qemu-kvm 是掌握 KVM 必须的最基础也是不可或缺的技能。其中的配置和编译 kvm（或 Linux kernel）是最复杂也是非常重要的环节，理解这环节对 Linux kernel 相关技术（包括 KVM）的学习具有非常重要的价值。

通过本章，相信你已经对 KVM 的虚拟化环境的构建比较了解了，请自己动手构建一套

自己的 KVM 虚拟化系统吧，也许在构建环境过程中你会碰到这样那样的问题，不过只有当你把它们都解决之后才能真正的理解。接下来，继续看后面的章节来学习 KVM 虚拟化都提供了那一些的功能、命令行以及它们的基本原理。

3.8 本章注释和参考阅读

1. 本章注释

- [1] KVM 在 ARM 处理器上第一次正式的实现，可以参考 <http://comments.gmane.org/gmane.comp.emulators.kvm.devel/87136> 中 主 题 为 “[PATCH v6 00/12] KVM/ARM Implementation” 的邮件。
- [2] 在本书中，若无特别注明，所有理论和实验都是基于 Intel x86-64 的处理器。
- [3] Tarball 是 UNIX/Linux 世界中的一个术语，用于指代 tar 档案（通过 tar 命令打包归档后的文件，如常见的以 .tar、.tar.gz、.tar.bz2 结尾的档案文件）。
- [4] Git 是一个高效、快速的分布式的版本控制工具和源代码管理工具，最初是 Linus Torvalds 为 Linux 内核开发而设计和开发的工具，是在 Linux 中最广泛使用的 SCM 工具。另外，一个基于 Git 的非常著名的源码托管网站 <https://github.com> 也值得推荐一下。
- [5] Linus Torvalds，从芬兰移民的美国人，著名的软件工程师和黑客。1991 年，在芬兰赫尔辛基大学读书的 Linus 发布了后来最流行的开源操作系统内核 Linux 的第一个版本。
- [6] 关于 QEMU 项目，可以参考其官方网站：http://wiki.qemu.org/Main_Page
- [7] 关于 QEMU 与 qemu-kvm 的关系：<http://wiki.qemu.org/KVM>。在 2012 年底，QEMU 的 1.3.0 版本发布时，qemu-kvm 中针对 KVM 的修改已经完全加入到普通的 QEMU 代码库中了，从此之后可以完全使用纯 QEMU 来与 KVM 配合使用（命令行添加 -enable-kvm 参数），而不是需要专门使用 qemu-kvm 代码库了。下载普通 QEMU 的代码仓库为：[git://git.qemu.org/qemu.git](https://git.qemu.org/qemu.git) 和 <http://git.qemu.org/git/qemu.git>。
- [8] 在宿主机中需要安装包含 vncserver 和 vncviewer 工具的软件包，如在 RHEL6.3 系统中，可以安装 tigervnc-server 和 tigervnc 这两个 RPM 软件包。

2. 参考阅读

- (1) KVM 官方网站的 FAQ（常见问题与答案）列表：<http://www.linux-kvm.org/page/FAQ>
- (2) KVM 宿主机或客户机安装 RHEL6.x 系统，也可参考如下链接中详细的英文文档：
http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Installation_Guide/index.html