Virtio：针对 Linux 的 I/O 虚拟化框架

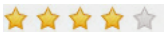
使用 KVM 和 lguest 的半虚拟化 I/O

M. Tim Jones, 自由作者

简介： Linux 内核支持多种虚拟化模式，并且支持的数量随着虚拟化的进步和新模式的出现（例如 `lguest`）而增加。但是，让这些虚拟化模式能够在 Linux 之上运行之后，又如何让它们能够在 I/O 虚拟化方面利用底层内核呢？答案是使用 `virtio`，它为 hypervisor 和一组通用的 I/O 虚拟化驱动程序提供高效的抽象。探索 virtio 并了解为什么 Linux 将成为最佳的 hypervisor。

分享您的技能： 支持特定的 I/O 虚拟化模式影响您选择使用指定的 hypervisor 吗？请在下面 添加您的评论。

发布日期： 2010 年 3 月 04 日
级别： 中级
其他语言版本： 英文
访问情况 ： 11342 次浏览
评论： 0 (查看 | 添加评论 - 登录)


平均分（16个评分）
为本文评分

联系 Tim

Tim 是我们最受欢迎并且很多产的作者之一。查看 Tim 的个人资料 并与他和 My developerWorks 上的其他作者和读者联系。

概而言之，`virtio` 是半虚拟化 hypervisor 中位于设备之上的抽象层。`virtio` 由 Rusty Russell 开发，他当时的目的是支持自己的虚拟化解决方案 `lguest`。本文在开篇时介绍半虚拟化和模拟设备，然后探索 `virtio` 的细节。本文的重点是来自 2.6.30 内核发行版的 `virtio` 框架。

Linux 是 hypervisor 展台。如我的 剖析 Linux hypervisor 所述，Linux 提供各种 hypervisor 解决方案，这些解决方案都有自己的特点和优点。这些解决方案包括 Kernel-based Virtual Machine（KVM）、`lguest` 和 User-mode Linux 等。在 Linux 上配备这些不同的 hypervisor 解决方案会给操作系统带来负担，负担的大小取决于各个解决方案的需求。其中的一项开销为设备的虚拟化。`virtio` 并没有提供多种设备模拟机制（针对网络、块和其他驱动程序），而是为这些设备模拟提供一个通用的前端，从而标准化接口和增加代码的跨平台重用。

完全虚拟化和半虚拟化

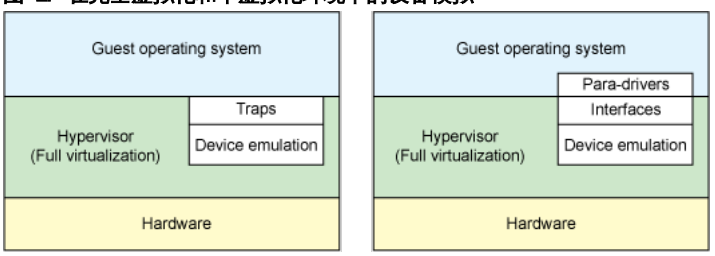
加入 My developerWorks 上的绿色小组

在 My developerWorks 上的 GReen IT Report 空间 和 绿色计算小组 上讨论关于能源、效率和环境的主题并共享资源。

让我们快速讨论一下两种类型完全不同的虚拟化模式：完全虚拟化和半虚拟化。在*完全虚拟化* 中，来宾操作系统运行在位于物理机器上的 hypervisor 之上。来宾操作系统并不知道它已被虚拟化，并且不需要任何更改就可以在该配置下工作。相反，在*半虚拟化* 中，来宾操作系统不仅知道它运行在 hypervisor 之上，还包含让来宾操作系统更高效地过渡到 hypervisor 的代码（见 图 1）。

在完全虚拟化模式中，hypervisor 必须模拟设备硬件，它是在会话的最低级别进行模拟的（例如，网络驱动程序）。尽管在该抽象中模拟很干净，但它同时也是最低效、最复杂的。在半虚拟化模式中，来宾操作系统和 hypervisor 能够共同合作，让模拟更加高效。半虚拟化方法的缺点是操作系统知道它被虚拟化，并且需要修改才能工作。

图 1. 在完全虚拟化和半虚拟化环境下的设备模拟



硬件随着虚拟化技术而不断改变。新的处理器通过纳入高级指令来让来宾操作系统到 hypervisor 的过渡更加高效。此外，硬件也随着输入/输出（I/O）虚拟化而不断改变（参见 参考资料 了解 Peripheral Controller Interconnect [PCI] passthrough 和 single- and multi-root I/O 虚拟化）。

virtio 的替换者

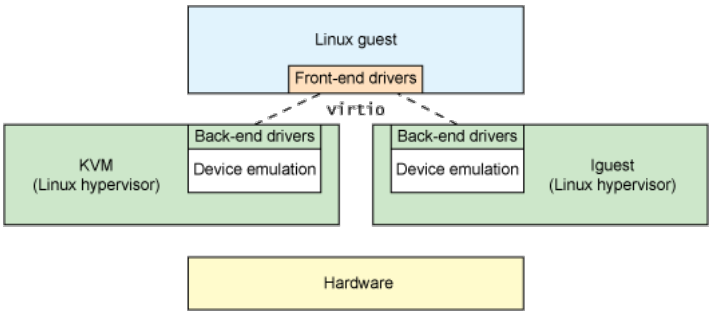
virtio 并不是该领域中的唯一霸主。Xen 提供半虚拟化设备驱动程序，VMware 也提供 *Guest Tools*。

但是在传统的完全虚拟化环境中，hypervisor 必须捕捉这些请求，然后模拟物理硬件的行为。尽管这样做提供很大的灵活性（即运行未更改的操作系统），但它的效率比较低（参见 图 1 左边）。图 1 的右边是半虚拟化示例。在这里，来宾操作系统知道它运行在 hypervisor 之上，并包含了充当前端的驱动程序。Hypervisor 为特定的设备模拟实现后端驱动程序。通过在这些前端和后端驱动程序中的 virtio，为开发模拟设备提供标准化接口，从而增加代码的跨平台重用率并提高效率。

针对 Linux 的抽象

从前面的小节可以看到，virtio 是对半虚拟化 hypervisor 中的一组通用模拟设备的抽象。该设置还允许 hypervisor 导出一组通用的模拟设备，并通过一个通用的应用编程接口（API）让它们变得可用。图 2 展示了为什么这很重要。有了半虚拟化 hypervisor 之后，来宾操作系统能够实现一组通用的接口，在一组后端驱动程序之后采用特定的设备模拟。后端驱动程序不需要是通用的，因为它们只实现前端所需的行为。

图 2. virtio 的驱动程序抽象



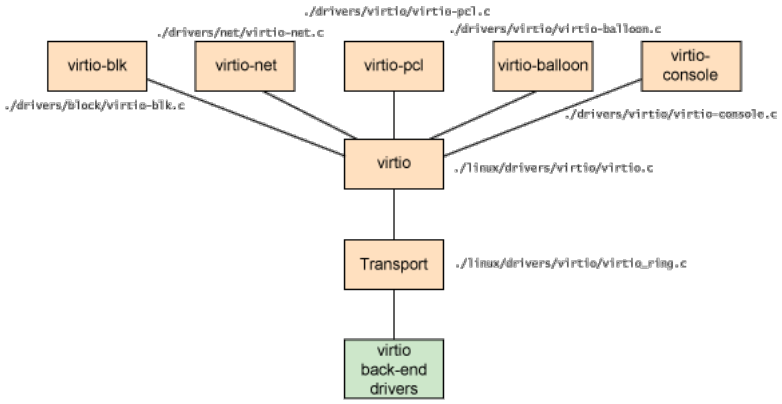
注意，在现实中（尽管不需要），设备模拟发生在使用 QEMU 的空间，因此后端驱动程序与 hypervisor 的用户空间交互，以通过 QEMU 为 I/O 提供便利。QEMU 是一个系统模拟器，它不仅提供来宾操作系统虚拟化平台，还提供整个系统（PCI 主机控制器、磁盘、网络、视频硬件、USB 控制器和其他硬件元素）的模拟。

virtio API 依赖一个简单的缓冲抽象来封装来宾操作系统需要的命令和数据。让我们查看 virtio API 的内部及其组件。

Virtio 架构

除了前端驱动程序（在来宾操作系统中实现）和后端驱动程序（在 hypervisor 中实现）之外，virtio 还定义了两个层来支持来宾操作系统到 hypervisor 的通信。在顶级（称为 *virtio*）的是虚拟队列接口，它在概念上将前端驱动程序附加到后端驱动程序。驱动程序可以使用 0 个或多个队列，具体数量取决于需求。例如，virtio 网络驱动程序使用两个虚拟队列（一个用于接收，另一个用于发送），而 virtio 块驱动程序仅使用一个虚拟队列。虚拟队列实际上被实现为跨越来宾操作系统和 hypervisor 的衔接点。但这可以通过任意方式实现，前提是来宾操作系统和 hypervisor 以相同的方式实现它。

图 3. vital 框架的高级架构

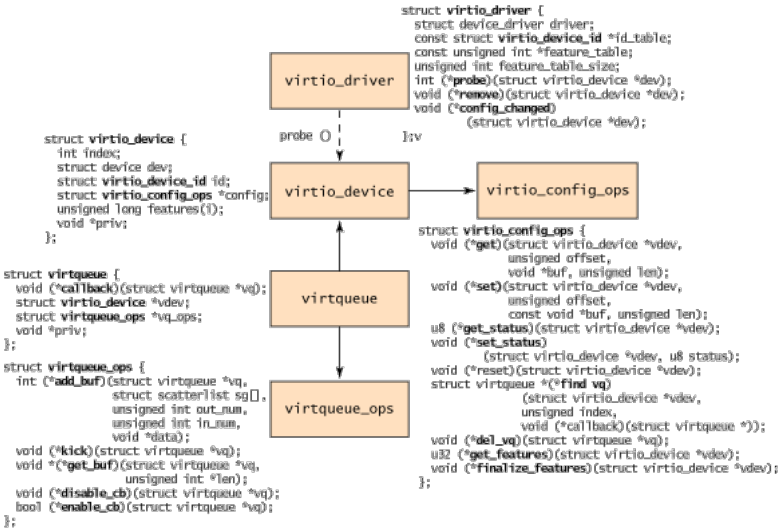


如 图 3 所示，分别为块设备（比如磁盘）、网络设备、PCI 模拟和 balloon 驱动程序列出了 5 个前端驱动程序。每个前端驱动程序在 hypervisor 中有一个对应的后端驱动程序。

概念层次结构

从来宾操作系统的角度来看，*对象层次结构* 的定义如 图 4 所示。在顶级的是 virtio_driver，它在来宾操作系统中表示前端驱动程序。与该驱动程序匹配的设备由 virtio_device（设备在来宾操作系统中的表示）封装。这引用 virtio_config_ops 结构（它定义配置 virtio 设备的操作）。virtio_device 由 virtqueue 引用（它包含一个到它服务的 virtio_device 的引用）。最后，每个 virtqueue 对象引用 virtqueue_ops 对象，后者定义处理 hypervisor 的驱动程序的底层队列操作。尽管队列操作是 virtio API 的核心，我还是先简单讨论一下新的发现，然后再详细探讨 virtqueue_ops 操作。

图 4. virtio 前端的对象层次结构



该流程以创建 `virtio_driver` 并通过 `register_virtio_driver` 进行注册开始。`virtio_driver` 结构定义上层设备驱动程序、驱动程序支持的设备 ID 的列表、一个特性表单（取决于设备类型）和一个回调函数列表。当 hypervisor 识别到与设备列表中的设备 ID 相匹配的新设备时，将调用 `probe` 函数（由 `virtio_driver` 对象提供）来传入 `virtio_device` 对象。将这个对象和设备的管理数据缓存起来（以独立于驱动程序的方式缓存）。可能要调用 `virtio_config_ops` 函数来获取或设置特定于设备的选项，例如，为 `virtio_blk` 设备获取磁盘的 Read/Write 状态或设置块设备的块大小，具体情况取决于启动器的类型。

注意，`virtio_device` 不包含到 `virtqueue` 的引用（但 `virtqueue` 确实引用了 `virtio_device`）。要识别与该 `virtio_device` 相关联的 `virtqueue`，您需要结合使用 `virtio_config_ops` 对象和 `find_vq` 函数。该对象返回与这个 `virtio_device` 实例相关联的虚拟队列。`find_vq` 函数还允许为 `virtqueue` 指定一个回调函数（查看 图 4 中的 `virtqueue` 结构）。

`virtqueue` 是一个简单的结构，它识别一个可选的回调函数（在 hypervisor 使用缓冲池时调用）、一个到 `virtio_device` 的引用、一个到 `virtqueue` 操作的引用，以及一个引用要使用的底层实现的特殊 `priv` 引用。虽然 `callback` 是可选的，但是它能够动态地启用或禁用回调。

该层次结构的核心是 `virtqueue_ops`，它定义在来宾操作系统和 hypervisor 之间移动命令和数据的方式。让我们首先探索添加到或从 `virtqueue` 移除的对象。

Virtio 缓冲池

来宾操作系统（前端）驱动程序通过缓冲池与 hypervisor 交互。对于 I/O，来宾操作系统提供一个或多个表示请求的缓冲池。例如，您可以提供 3 个缓冲池，第一个表示 Read 请求，后面两个表示响应数据。该配置在内部被表示为一个散集列表（scatter-gather），列表中的每个条目表示一个地址和一个长度。

核心 API

通过 `virtio_device` 和 `virtqueue`（更常见）将来宾操作系统驱动程序与 hypervisor 的驱动程序链接起来。`virtqueue` 支持它自己的由 5 个函数组成的 API。您可以使用第一个函数 `add_buf` 来向 hypervisor 提供请求。如前面所述，该请求以散集列表的形式存在。对于 `add_buf`，来宾操作系统提供用于将请求添加到队列的 `virtqueue`、散集列表（地址和长度数组）、用作输出条目（目标是底层 hypervisor）的缓冲池数量，以及用作输入条目（hypervisor 将为它们储存数据并返回到来宾操作系统）的缓冲池数量。当通过 `add_buf` 向 hypervisor 发出请求时，来宾操作系统能够通过 `kick` 函数通知 hypervisor 新的请求。为了获得最佳的性能，来宾操作系统应该在通过 `kick` 发出通知之前将尽可能多的缓冲池装载到 `virtqueue`。

通过 `get_buf` 函数触发来自 hypervisor 的响应。来宾操作系统仅需调用该函数或通过提供的 `virtqueue callback` 函数等待通知就可以实现轮询。当来宾操作系统知道缓冲区可用时，调用 `get_buf` 返回完成的缓冲区。

`virtqueue` API 的最后两个函数是 `enable_cb` 和 `disable_cb`。您可以使用这两个函数来启用或禁用回调进程（通过在 `virtqueue` 中由 `virtqueue` 初始化的 `callback` 函数）。注意，该回调函数和 hypervisor 位于独立的地址空间中，因此调用通过一个间接的 hypervisor 来触发（比如 `kvm_hypercall`）。

缓冲区的格式、顺序和内容仅对前端和后端驱动程序有意义。内部传输（当前实现中的连接点）仅移动缓冲区，并且不知道它们的内部表示。

示例 virtio 驱动程序

您可以在 Linux 内核的 ./drivers 子目录内找到各种前端驱动程序的源代码。可以在 ./drivers/net/virtio_ net.c 中找到 `virtio` 网络驱动程序，在 ./drivers/block/virtio_ blk.c 中找到 `virtio` 块驱动程序。子目录 ./drivers/virtio 提供 `virtio` 接口的实现（`virtio` 设备、驱动程序、`virtqueue` 和连接点）。`virtio` 还应用在 High-Performance Computing (HPC) 研究中，以开发出通过共享内存传递的 inter-virtual machine (VM) 通信。尤其是，这是通过使用 `virtio` PCI 驱动程序的虚拟化 PCI 接口实现的。您可以在 参考资料 部分更多地了解这个知识点。

现在，您可以在 Linux 内核中实践这个半虚拟化基础架构。您所需的包括一个充当 hypervisor 的内核、一个来宾操作性内核和用于设备模拟的 QEMU。您可以使用 KVM（位于主机内核中的一个模块）或 Rusty Russell 的 `lguest`（修改版的 Linux 来宾操作系统内核）。这两个虚拟化解决方案都支持 `virtio`（以及用于系统模拟的 QEMU 和用于虚拟化管理的 `libvirt`）。

Rusty 的 lguest 是针对半虚拟化驱动程序和更快速地模拟虚拟设备的更简洁代码库。但更重要的是，实践证明 virtio 比现有的商业解决方案提供更出色的性能（网络 I/O 能够提升 2-3 倍）。性能的提升是需要付出代价的，但是如果您使用 Linux 作为 hypervisor 和来宾操作系统，那么这样做是值得的。

结束语

也许您可能从来没有为 virtio 开发过前端或后端驱动程序，它实现了一个有趣的架构，值得您仔细去探索。virtio 为提高半虚拟化 I/O 环境中的效率带来了新的机会，同时能够利用 Xen 以前的成果。Linux 不断地证明它是一个产品 hypervisor，并且是新虚拟化技术研究平台。virtio 这个例子展示了将 Linux 用作 hypervisor 的强大之处和开放性。

参考资料

学习

- Rusty Russell 的“ Virtio: towards a de factor standard for virtual I/O devices” 是深入了解 virtio 技术的最好资源。这篇论文详尽阐述了 virtio 及其内部结构。
- 这篇文章谈论两个虚拟化机制：完全虚拟化和半虚拟化。要更多地了解 Linux 中的各种虚拟化机制，请查看 Tim 的文章“ 虚拟 Linux”（developerWorks，2006 年 12 月）。
- virtio 背后的秘密就是利用半虚拟化来改善总体 I/O 性能。要了解使用 Linux 作为 hypervisor 和设备模拟，请查看 Tim 的文章“ 剖析 Linux hypervisor”（developerWorks，2009 年 5 月）和“ Linux 虚拟化和 PCI 透传技术”（developerWorks，2009 年 10 月）。
- 本文讨论设备模拟，提供该功能的最重要应用程序之一是 QEMU（一个系统模拟器）。您可以阅读 Tim 的文章“ 使用 QEMU 进行系统仿真”（developerWorks，2007 年 9 月），更多地了解 QEMU。
- Xen 还包含虚拟化驱动程序的概念。Paravirtual Windows Drivers 讨论半虚拟化和硬件辅助虚拟化（HVM），尤其是后者。
- virtio 的最重要优点之一是在半虚拟化环境中提升效率。这篇来自 btm.geek 的博客显示了 使用 KVM 的 virtio 的优势。
- 本文讨论 libvirt（一个开源虚拟化 API）和 virtio 框架的相似之处。libvirt wiki 展示了如何在 libvirt 中指定 virtio 设备。
- 这篇文章讨论两个利用 virtio 框架的 hypervisor 解决方案：lguest 是一个 x86 hypervisor，它也是由 Rusty Russell 开发的；KVM 是另一个基于 Linux 的 hypervisor，它是首个构建到 Linux 内核中的 hypervisor。
- virtio 的有趣应用之一是开发 共享内存消息传递，以让 VM 能够通过 hypervisor 彼此通信，来自 SpringerLink 的论文对此进行了阐述。
- 在 developerWorks Linux 专区 寻找为 Linux 开发人员（包括 Linux 新手入门）准备的更多参考资料，查阅我们 最受欢迎的文章和教程。
- 在 developerWorks 上查阅所有 Linux 技巧 和 Linux 教程。
- 随时关注 developerWorks 技术活动和网络广播。

获得产品和技术

- 使用可直接从 developerWorks 下载的 IBM 产品评估试用版软件 构建您的下一个 Linux 开发项目。

讨论

- 加入 My developerWorks 社区。查看开发人员参与的博客、论坛、组和 wiki，并与其他 developerWorks 用户交流。

关于作者

M. Tim Jones 是一名嵌入式固件架构师，同时也是 *Artificial Intelligence: A Systems Approach*, *GNU/Linux Application Programming*（第二版）、*AI Application Programming*（第二版）和 *BSD Sockets Programming from a Multilanguage Perspective* 的作者。他的工程背景包括地球同步航天器内核开发、嵌入式系统架构和网络协议开发等。Tim 还是科罗拉多州朗蒙特市 Emulex Corp. 的顾问工程师。

关闭 [x]

developerWorks：登录

IBM ID：
需要一个 IBM ID？
忘记 IBM ID？

密码：
忘记密码？
更改您的密码

保持登录。

单击提交则表示您同意developerWorks 的条款和条件。 使用条款

提交 取消

当您初次登录到 developerWorks 时，将会为您创建一份概要信息。**您在 developerWorks 概要信息中选择公开的信息将公开显示给其他人，但您可以随时修改这些信息的显示状态。**您的姓名（除非选择隐藏）和昵称将和您在 developerWorks 发布的内容一同显示。

所有提交的信息确保安全。

关闭 [x]

请选择您的昵称：

当您初次登录到 developerWorks 时，将会为您创建一份概要信息，您需要指定一个昵称。您的昵称将和您在 developerWorks 发布的内容显示在一起。

昵称长度在 3 至 31 个字符之间。 您的昵称在 developerWorks 社区中必须是唯一的，并且出于隐私保护的原因，不能是您的电子邮件地址。

昵称： （长度在 3 至 31 个字符之间）

单击**提交**则表示您同意developerWorks 的条款和条件。 使用条款.

提交 取消

所有提交的信息确保安全。

平均分（16个评分）

1 星

1 星
2 星

2 星
3 星

3 星
4 星

4 星
5 星

5 星

提交

添加评论:

请 登录 或 注册 后发表评论。

注意：评论中不支持 HTML 语法

有新评论时提醒我剩余 1000 字符

发布

快来添加第一条评论