导航

博客园
首页
新随笔
联系
订阅 XML
管理

< 2020年4月 >

日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

公告

昵称： 钱小小
园龄： 5年5个月
粉丝： 0
关注： 1
+加关注

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

linux基础(3)
网络(3)
转载(3)
GRO(2)
linux内核(2)
nova(1)
open-falcon(1)
OpenStack(1)
iptables(1)
jsonrpc(1)
更多

【翻译】QEMU内部机制：内存

系列文章：

1. 【翻译】QEMU内部机制：宏观架构和线程模型
2. 【翻译】QEMU内部机制：vhost的架构
3. 【翻译】QEMU内部机制：顶层概览
4. 【翻译】QEMU内部机制：内存(本文)

原文地址：https://blog.vmsplice.net/2016/01/qemu-internals-how-guest-physical-ram.html

原文时间：2016年1月8日

作者介绍：Stefan Hajnoczi来自红帽公司的虚拟化团队，负责开发和维护QEMU项目的block layer, network subsystem和tracing subsystem。

目前工作是multi-core device emulation in QEMU和host/guest file sharing using vsock，过去从事过disk image formats, storage migration和I/O performance optimization

```
什么是DIMM?
在80286时代，内存颗粒（Chip）是直接插在主板上的，叫做DIP(Dual In-line Package)。
到了80386时代，换成1片焊有内存颗粒的电路板，叫做SIMM（Single-Inline Memory Module）。
由阵脚形态变化成电路板带来了很多好处:模块化，安装便利等等，由此DIY市场才有可能产生。
当时SIMM的位宽是32bit，即一个周期读取4个字节，到了奔腾时，位宽变为64bit，即8个字节，于是SIMM就顺势变为DIMM（Double-Inline Memory Module）。
这种形态一直延续至今，也是内存条的基本形态。
内存和RAM?
内存一般采用半导体存储单元，包括随机存储器（RAM），只读存储器（ROM），以及高速缓存（CACHE）。
只不过因为RAM是其中最重要的存储器。 通常所说的内存即指电脑系统中的RAM。RAM要求每时每刻都不断地供电，否则数据会丢失
```

vm的物理内存是如何工作的

随笔分类

Apache全景分析(1)
QT(1)
安全
编程基础(27)
技术调研(19)
数据库(1)
虚拟化(13)

随笔档案

2020年2月(1)
2019年7月(14)
2019年6月(16)
2019年5月(1)
2019年4月(6)
2016年3月(2)
2015年12月(2)
2015年3月(12)
2014年12月(2)
2014年11月(9)

wxWidgets相关

wxWidgets简明教程
wxWidgets的linux安装配置
wxWidgets在线帮助

最新评论

1. Re:【转】extern "C"的含义和用法
nm test.so可以查看符号表
--钱小小
2. Re:"云端融合"思想的自我摸索（很不靠谱）
目前，云计算技术的发展日新月异，它可以促进信息技术和数据资源充分合理利用，是信息化发展的必然趋势。下面是我对于云计算与操作系统的简单想法，其中内容不乏失实及误导之陈述，请领导批评指正。 经过将近十年的...
--钱小小
3. Re:debian包的补丁管理工具：quilt
来自ubuntu的dh_quilt_patch命令帮助: NAME dh_quilt_patch - apply patches listed in debian/patches/seriesSYNOP...
--钱小小

内存是模拟系统中的关键概念。QEMU使用协同工作的多个组件对vm的内存进行建模。
本文从宏观角度讨论QEMU2.5中的vm物理内存的设计，不会涉及过多细节。读完本文后，你就可以去读qemu项目的源码了。

注意这里并不会讨论vm的虚拟内存，它可能在另一篇文章中讨论。kvm虚拟化依赖于硬件内存翻译的支持，它不使用QEMU的软件实现的内存管理单元(MMU)。

vm RAM的配置

QEMU命令行选项-m [size=]megs[,slots=n,maxmem=size]用于设置vm的初始RAM大小、vm RAM的最大值和内存条的插槽数(DIMM:双列直插式内存模块)。
能够配置内存最大值、插槽数的原因是，QEMU利用类似于物理机发现新内存或移除内存的机制模拟内存条的热插拔。包括类似于物理机的插槽，插入或拔出内存条。也就是说，更改可得内存并不是以字节为单位的，而是以插入到vm中的内存条数量为单位的。

vm内存的热插拔性

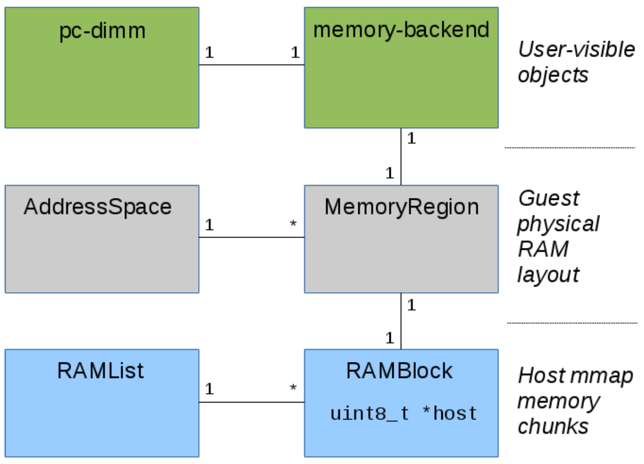
pc-dimm设备(hw/mem/pc-dimm.c)用来表示一个内存条。内存可以通过创建一个新的pc-dimm设备实现热插功能。虽然名称中有pc字段，但该设备也可与ppc、s390机器配合使用。
要注意的是，vm启动时的初始RAM可能无法使用"pc-dimm"设备建模，并且无法热拔出。
vm RAM本身并不在pc-dimm对象的容器中，也就是说，pc-dimm对象必须与内存后端对象关联。

内存后端

内存后端设备(backends/hostmem.c)中包含了真实的宿主机内存，它被用于支持vm RAM的实现。它可以是匿名mmapped内存或基于文件的mmapped内存。基于文件的vm RAM允许为宿主机上的大页面使用hugetlbfs，同时也允许使用共享内存，以便其他宿主机应用程序可以访问vm的RAM。

pc-dimm和内存后端对象是QEMU中用户视角的vm RAM部分。他们可以通过命令行或者QMP监视器接口进行管理。这仅仅是冰山一角，下面将介绍QEMU中其他几个vm RAM的内部机制。

下图是这几种组件的关系图：



RAM块和ram_addr_t地址空间

内存后端中的内存实际上是由RAMBlock通过qemu_ram_alloc()函数(exec.c)分配(mmapped)的。每一个RAMBlock对象都有一个指向mmap内存的指针和一个ram_addr_t偏移量。ram_addr_t偏移量比较有意思，因为它为了实现RAMBlock的查找而位于全局命名空间中。

ram_addr_t命名空间与vm物理内存空间不同，ram_addr_t命名空间是所有RAMBlock的紧密堆积地址空间(什么意思？)。vm物理地址的0x100001000与ram_addr_t的0x100001000可能是不同的，因为ram_addr_t不包括保留的vm物理内存区域、内存映射I/O等。而且，ram_addr_t偏移量依赖于RAMBlocks被建立的次序，不像vm物理内存空间中都有一个固定位置。

所有的RAMBlocks对象都位于一个RAMList类型的对象ram_list中，它保存有RAMBlocks对象和脏内存位图(dirty memory bitmaps)。

跟踪脏内存

当vm的cpu或设备DMA向vm RAM写入时，下列几个"用户"需要得知此事件：
1.动态迁移特性依赖于脏内存页面追踪机制，以便在其更改是可以重新发送至目的端。
2.TCG依赖于跟踪自修改代码，以便重新编译已更改的指令
3.图形卡仿真依赖于跟踪脏视频内存以便仅对已更改的扫描线进行重绘
在ram_list对象中包含多个脏内存位图，他们可以根据上述"用户"的需求分别启用或禁用。

4. Re:在Linux下开发多语言软件(gettext解决方案)
附图bug修复报告：
--钱小小

阅读排行榜

1. golang类型断言的使用（Type Assertion）(3770)
2. 进程占用过高cpu的排查(977)
3. 【转】QEMU Monitor机制实例分析(792)
4. 【转】理解qemu对设备的模拟机制(618)
5. 【翻译】QEMU内部机制：宏观架构和线程模型(402)

评论排行榜

1. 在Linux下开发多语言软件(gettext解决方案)(1)
2. debian包的补丁管理工具：quilt(1)
3. “云端融合”思想的自我摸索（很不靠谱）(1)
4. 【转】extern "C"的含义和用法(1)

地址空间

所有的cpu架构都包含一个内存空间，有些还包含一块IO地址空间。QEMU使用AddressSpace表示这些空间，其中包含了一棵MemoryRegions对象的树结构(见include/exec/memory.h)。

MemoryRegion对象是连接vm物理地址空间和包含有内存的RAMBlocks对象的纽带。MemoryRegion对象中包含RAMBlock的ram_addr_t偏移量，而RAMBlock对象包含有指向MemoryRegion的指针。

注意，MemoryRegion对象的概念比RAM要广泛，它还可以表示在访问时调用读/写回调函数的I/O内存。这就是把来自vm CPU的硬件寄存器访问分派到仿真设备的方式。

address_space_rw()函数将读取/存储的访问请求分发到正确的MemoryRegion对象，如果MemoryRegion是RAM区域，那么将从RAMBlock的mmapped的vm RAM中访问数据。address_space_memory全局变量表示vm物理内存空间。

结论

管理vm物理内存的机制有很多层次。pc-dimm和内存后端对象是用户可见的DIMM和内存配置对象，RAMBlock是mmapped的内存块。AddressSpace和MemoryRegion对象将vm的RAM放置到内存映射中。

原文如下：

QEMU Internals: How guest physical RAM works

Memory is one of the key aspects of emulating computer systems. Inside QEMU the guest RAM is modelled by several components that work together. This post gives an overview of the design of guest physical RAM in QEMU 2.5 by explaining the most important components without going into all the details. After reading this post you will know enough to dig into the QEMU source code yourself.

Note that guest virtual memory is not covered here since it deserves its own post. KVM virtualization relies on hardware memory translation support and does not use QEMU's software MMU.

Guest RAM configuration

The QEMU command-line option `-m [size=]megs[,slots=n,maxmem=size]` specifies the initial guest RAM size as well as the maximum guest RAM size and number of slots for memory chips (DIMMs).

The reason for the maximum size and slots is that QEMU emulates DIMM hotplug so the guest operating system can detect when new memory is added and removed using the same mechanism as on real hardware. This involves plugging or unplugging a DIMM into a slot, just like on a physical machine. In other words, changing the amount of memory available isn't done in byte units, it's done by changing the set of DIMMs plugged into the emulated machine.

Hotpluggable guest memory

The "pc-dimm" device (hw/mem/pc-dimm.c) models a DIMM. Memory is hotplugged by creating a new "pc-dimm" device. Although the name includes "pc" this device is also used with ppc and s390 machine types.

As a side-note, the initial RAM that the guest started with might not be modelled with a "pc-dimm" device and it can't be unplugged.

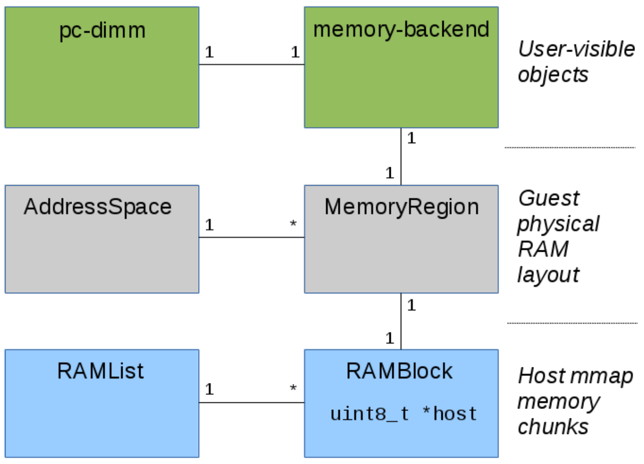
The guest RAM itself isn't contained inside the "pc-dimm" object. Instead the "pc-dimm" must be associated with a "memory-backend" object.

Memory backends

The "memory-backend" device (backends/hostmem.c) contains the actual host memory that backs guest RAM. This can either be anonymous mmapped memory or file-backed mmapped memory. File-backed guest RAM allows Linux hugetlbfs usage for huge pages on the host and also shared-memory so other host applications can access to guest RAM.

The "pc-dimm" and "memory-backend" objects are the user-visible parts of guest RAM in QEMU. They can be managed using the QEMU command-line and QMP monitor interface. This is just the tip of the iceberg though because there are still several aspects of guest RAM internal to QEMU that will be covered next.

The following diagram shows the components explained below:



RAM blocks and the ram_addr_t address space

Memory inside a "memory-backend" is actually mmapped by RAMBlock through qemu_ram_alloc() (exec.c). Each RAMBlock has a pointer to the mmap memory and also a ram_addr_t offset. This ram_addr_t offset is interesting because it is in a global namespace so the RAMBlock can be looked up by the offset.

The ram_addr_t namespace is different from the guest physical memory space. The ram_addr_t namespace is a tightly packed address space of all RAMBlocks. Guest physical address 0x100001000 might not be ram_addr_t 0x100001000 since ram_addr_t does not include guest physical memory regions that are reserved, memory-mapped I/O, etc. Furthermore, the ram_addr_t offset is dependent on the order in which RAMBlocks were created, unlike the guest physical memory space where everything has a fixed location.

All RAMBlocks are in a global list RAMList object called ram_list. ram_list holds the RAMBlocks and also the dirty memory bitmaps.

Dirty memory tracking

When the guest CPU or device DMA stores to guest RAM this needs to be noticed by several users:

1. The live migration feature relies on tracking dirty memory pages so they can be resent if they change during live migration.
2. TCG relies on tracking self-modifying code so it can recompile changed instructions.
3. Graphics card emulation relies on tracking dirty video memory to redraw only scanlines that have changed.

There are dirty memory bitmaps for each of these users in ram_list because dirty memory tracking can be enabled or disabled independently for each of these users.

Address spaces

All CPU architectures have a memory space and some also have an I/O address space. This is represented by AddressSpace, which contains a tree of MemoryRegions (include/exec/memory.h).

The MemoryRegion is the link between guest physical address space and the RAMBlocks containing the memory. Each MemoryRegion has the ram_addr_t offset of the RAMBlock and each RAMBlock has a MemoryRegion pointer.

Note that MemoryRegion is more general than just RAM. It can also represent I/O memory where read/write callback functions are invoked on access. This is how hardware register accesses from a guest CPU are dispatched to emulated devices.

The `address_space_rw()` function dispatches load/store accesses to the appropriate MemoryRegions. If a MemoryRegion is a RAM region then the data will be accessed from the RAMBlock's mmapped guest RAM.

The`address_space_memory` global variable is the guest physical memory space.

Conclusion

There are a few different layers involved in managing guest physical memory. The "pc-dimm" and "memory-backend" objects are the user-visible configuration objects for DIMMs and memory. The RAMBlock is the mmapped memory chunk. The AddressSpace and its MemoryRegion elements place guest RAM into the memory map.

« 上一篇： 进程占用过高cpu的排查
» 下一篇： 【转】Garbage collection in Python: things you need to know

相关博文：

- Linux设备驱动--内存管理
- mmap内存映射学习笔记
- linux内存管理机制（自己整理）
- 计算机支持的最大内存与CPU之间的关系
- 【嵌入式】内存管理，虚拟存储

» 更多推荐...

最新 IT 新闻:

- 特斯拉经营范围新增电信业务等 并正式迁入上海自贸区
- 任正非最新讲话：艰苦奋斗的目的是过幸福生活
- 聚美优品宣布完成私有化，正式从纽交所退市
- 智能高空作业机器人公司史河科技完成3500万元Pre A+轮融资
- 亚马逊下调广告营销联盟佣金费率 或重创出版商营收

» 更多新闻...