Stefan Hajnoczi

Open source and virtualization blog

Friday, January 8, 2016

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.

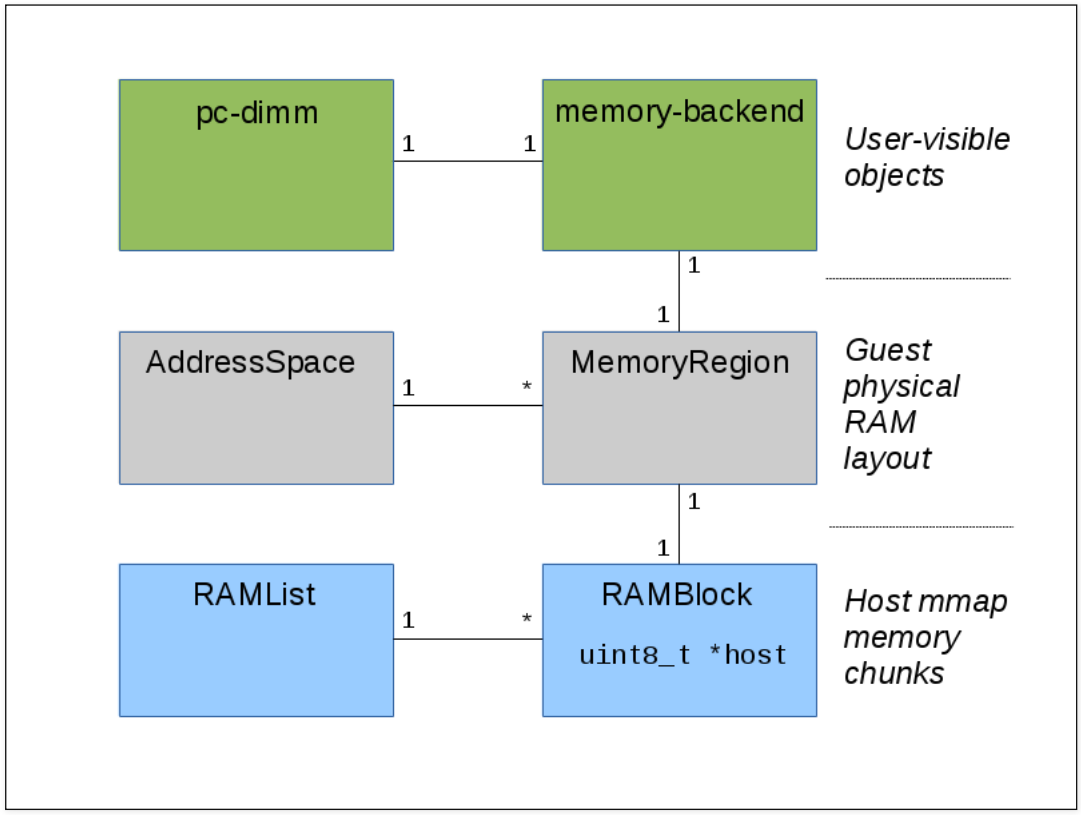
The "pc-dimm" and "memory-backend" objects are the user-visible parts of guest RAM in QEMU. They can be managed using the QEMU command-line and QMP monitor interface. This is just the tip of the iceberg though because there are still several aspects of guest RAM internal to QEMU that will be covered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

Blog Archive
► 2023
► 2022
► 2021
► 2020
► 2019
► 2018
► 2017
▼ 2016
► De
► Se
► Ma
► Ma
▼ Jar
QEI
H
ph
w
► 2015
► 2014
► 2013
► 2012
► 2011

Labels
- block
- flatpak
- git
- gsoc
- interna
- kvm
- libvirt
- linux
- nbd
- qemu
- tracing
- virtio

About me

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

I am an o
developer
contribute
and KVM
Hat. The
this site a
and don't
represent
positions,
opinions.

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.

Posted by stefanha at 9:24 AM

Newer Post Home Older Post

Have a question?

Please email qemu-devel@nongnu.org and CC stefanha@gmail.com with your question about this blog post. This way ot participate in the discussion on the QEMU mailing list.