The x86 kvm shadow mmu

The mmu (in arch/x86/kvm, files mmu.[ch] and paging_tmpl.h) is responsible for presenting a standard x86 mmu to the guest, while translating guest physical addresses to host physical addresses.

The mmu code attempts to satisfy the following requirements:

- correctness: the guest should not be able to determine that it is running on an emulated mmu except for timing (we attempt to comply with the specification, not emulate the characteristics of a particular implementation such as tlb size)
- security: the guest must not be able to touch host memory not assigned to it
- performance: minimize the performance penalty imposed by the mmu
- scaling: need to scale to large memory and large vcpu guests
- hardware: support the full range of x86 virtualization hardware
- integration: Linux memory management code must be in control of guest memory so that swapping, page migration, page merging, transparent hugepages, and similar features work without change
- dirty tracking: report writes to guest memory to enable live migration and framebuffer-based displays
- footprint: keep the amount of pinned kernel memory low (most memory should be shrinkable)
- reliablity: avoid multipage or GFP_ATOMIC allocations

Acronyms

pfn	host page frame number
hpa	host physical address
hva	host virtual address
gfn	guest frame number
gpa	guest physical address
gva	guest virtual address
ngpa	nested guest physical address
ngva	nested guest virtual address
pte	page table entry (used also to refer generically to paging structure entries)
gpte	guest pte (referring to gfns)
spte	shadow pte (referring to pfns)
tdp	two dimensional paging (vendor neutral term for NPT and EPT)

Virtual and real hardware supported

The mmu supports first-generation mmu hardware, which allows an atomic switch of the current paging mode and cr3 during guest entry, as well as two-dimensional paging (AMD's NPT and Intel's EPT). The emulated hardware it exposes is the traditional 2/3/4 level x86 mmu, with support for global pages, pae, pse, pse36, cr0.wp, and 1GB pages. Work is in progress to support exposing NPT capable hardware on NPT capable hosts.

Translation

The primary job of the mmu is to program the processor's mmu to translate addresses for the guest. Different translations are required at different times:

- when guest paging is disabled, we translate guest physical addresses to host physical addresses (gpa->hpa)
- when guest paging is enabled, we translate guest virtual addresses, to guest physical addresses, to host physical addresses (gva->gpa->hpa)
- when the guest launches a guest of its own, we translate nested guest virtual addresses, to nested guest physical addresses, to guest physical addresses, to host physical addresses (ngva->ngpa->gpa->hpa)

The primary challenge is to encode between 1 and 3 translations into hardware that support only 1 (traditional) and 2 (tdp) translations. When the number of required translations matches the hardware, the mmu operates in direct mode; otherwise it operates in shadow mode (see below).

Memory
======

Guest memory (gpa) is part of the user address space of the process that is using kvm. Userspace defines the translation between guest addresses and user addresses (gpa->hva); note that two gpas may alias to the same gva, but not vice versa.

These gvas may be backed using any method available to the host: anonymous memory, file backed memory, and device memory. Memory might be paged by the host at any time.

Events
======

The mmu is driven by events, some from the guest, some from the host.

Guest generated events:
- writes to control registers (especially cr3)
- invlpg/invlpga instruction execution
- access to missing or protected translations

Host generated events:
- changes in the gpa->hpa translation (either through gpa->hva changes or through hva->hpa changes)
- memory pressure (the shrinker)

Shadow pages
============

The principal data structure is the shadow page, 'struct kvm_mmu_page'. A shadow page contains 512 sptes, which can be either leaf or nonleaf sptes. A shadow page may contain a mix of leaf and nonleaf sptes.

A nonleaf spte allows the hardware mmu to reach the leaf pages and is not related to a translation directly. It points to other shadow pages.

A leaf spte corresponds to either one or two translations encoded into

one paging structure entry. These are always the lowest level of the
translation stack, with optional higher level translations left to NPT/EPT.
Leaf ptes point at guest pages.

The following table shows translations encoded by leaf ptes, with higher-level
translations in parentheses:

```
 Non-nested guests:
  nonpaging:      gpa->hpa
  paging:         gva->gpa->hpa
  paging, tdp:    (gva->)gpa->hpa
 Nested guests:
  non-tdp:        ngva->gpa->hpa  (*)
  tdp:            (ngva->)ngpa->gpa->hpa
```

(*) the guest hypervisor will encode the ngva->gpa translation into its page
    tables if npt is not present

Shadow pages contain the following information:
  role.level:
    The level in the shadow paging hierarchy that this shadow page belongs to.
    1=4k sptes, 2=2M sptes, 3=1G sptes, etc.
  role.direct:
    If set, leaf sptes reachable from this page are for a linear range.
    Examples include real mode translation, large guest pages backed by small
    host pages, and gpa->hpa translations when NPT or EPT is active.
    The linear range starts at (gfn << PAGE_SHIFT) and its size is determined
    by role.level (2MB for first level, 1GB for second level, 0.5TB for third
    level, 256TB for fourth level)
    If clear, this page corresponds to a guest page table denoted by the gfn
    field.
  role.quadrant:
    When role.cr4_pae=0, the guest uses 32-bit gptes while the host uses 64-bit
    sptes. That means a guest page table contains more ptes than the host,
    so multiple shadow pages are needed to shadow one guest page.
    For first-level shadow pages, role.quadrant can be 0 or 1 and denotes the
    first or second 512-gpte block in the guest page table. For second-level
    page tables, each 32-bit gpte is converted to two 64-bit sptes
    (since each first-level guest page is shadowed by two first-level
    shadow pages) so role.quadrant takes values in the range 0..3. Each
    quadrant maps 1GB virtual address space.
  role.access:
    Inherited guest access permissions in the form uwx. Note execute
    permission is positive, not negative.
  role.invalid:
    The page is invalid and should not be used. It is a root page that is
    currently pinned (by a cpu hardware register pointing to it); once it is
    unpinned it will be destroyed.
  role.cr4_pae:
    Contains the value of cr4.pae for which the page is valid (e.g. whether
    32-bit or 64-bit gptes are in use).
  role.cr4_nxe:
    Contains the value of efer.nxe for which the page is valid.
  role.cr0_wp:
    Contains the value of cr0.wp for which the page is valid.
  gfn:

  Either the guest page table containing the translations shadowed by this
  page, or the base page frame for linear translations. See role.direct.
spt:
  A pageful of 64-bit sptes containing the translations for this page.
  Accessed by both kvm and hardware.
  The page pointed to by spt will have its page->private pointing back
  at the shadow page structure.
  sptes in spt point either at guest pages, or at lower-level shadow pages.
  Specifically, if sp1 and sp2 are shadow pages, then sp1->spt[n] may point
  at __pa(sp2->spt). sp2 will point back at sp1 through parent_pte.
  The spt array forms a DAG structure with the shadow page as a node, and
  guest pages as leaves.
gfns:
  An array of 512 guest frame numbers, one for each present pte. Used to
  perform a reverse map from a pte to a gfn.
slot_bitmap:
  A bitmap containing one bit per memory slot. If the page contains a pte
  mapping a page from memory slot n, then bit n of slot_bitmap will be set
  (if a page is aliased among several slots, then it is not guaranteed that
  all slots will be marked).
  Used during dirty logging to avoid scanning a shadow page if none if its
  pages need tracking.
root_count:
  A counter keeping track of how many hardware registers (guest cr3 or
  pdptrs) are now pointing at the page. While this counter is nonzero, the
  page cannot be destroyed. See role.invalid.
multimapped:
  Whether there exist multiple sptes pointing at this page.
parent_pte/parent_ptes:
  If multimapped is zero, parent_pte points at the single spte that points at
  this page's spt. Otherwise, parent_ptes points at a data structure
  with a list of parent_ptes.
unsync:
  If true, then the translations in this page may not match the guest's
  translation. This is equivalent to the state of the tlb when a pte is
  changed but before the tlb entry is flushed. Accordingly, unsync ptes
  are synchronized when the guest executes invlpg or flushes its tlb by
  other means. Valid for leaf pages.
unsync_children:
  How many sptes in the page point at pages that are unsync (or have
  unsynchronized children).
unsync_child_bitmap:
  A bitmap indicating which sptes in spt point (directly or indirectly) at
  pages that may be unsynchronized. Used to quickly locate all unsychronized
  pages reachable from a given page.

Reverse map
===========

The mmu maintains a reverse mapping whereby all ptes mapping a page can be
reached given its gfn. This is used, for example, when swapping out a page.

Synchronized and unsynchronized pages
=====================================

The guest uses two events to synchronize its tlb and page tables: tlb flushes

and page invalidations (invlpg).

A tlb flush means that we need to synchronize all sptes reachable from the guest's cr3. This is expensive, so we keep all guest page tables write protected, and synchronize sptes to gptes when a gpte is written.

A special case is when a guest page table is reachable from the current guest cr3. In this case, the guest is obliged to issue an invlpg instruction before using the translation. We take advantage of that by removing write protection from the guest page, and allowing the guest to modify it freely. We synchronize modified gptes when the guest invokes invlpg. This reduces the amount of emulation we have to do when the guest modifies multiple gptes, or when the a guest page is no longer used as a page table and is used for random guest data.

As a side effect we have to resynchronize all reachable unsynchronized shadow pages on a tlb flush.

Reaction to events
==================

- guest page fault (or npt page fault, or ept violation)

This is the most complicated event. The cause of a page fault can be:

  - a true guest fault (the guest translation won't allow the access) (*)
  - access to a missing translation
  - access to a protected translation
    - when logging dirty pages, memory is write protected
    - synchronized shadow pages are write protected (*)
  - access to untranslatable memory (mmio)

  (*) not applicable in direct mode

Handling a page fault is performed as follows:

 - if needed, walk the guest page tables to determine the guest translation
   (gva->gpa or ngpa->gpa)
   - if permissions are insufficient, reflect the fault back to the guest
 - determine the host page
   - if this is an mmio request, there is no host page; call the emulator
     to emulate the instruction instead
 - walk the shadow page table to find the spte for the translation,
   instantiating missing intermediate page tables as necessary
 - try to unsynchronize the page
   - if successful, we can let the guest continue and modify the gpte
 - emulate the instruction
   - if failed, unshadow the page and let the guest continue
 - update any translations that were modified by the instruction

invlpg handling:

  - walk the shadow page hierarchy and drop affected translations
  - try to reinstantiate the indicated translation in the hope that the
    guest will use it in the near future

Guest control register updates:

- mov to cr3
  - look up new shadow roots
  - synchronize newly reachable shadow pages

- mov to cr0/cr4/efer
  - set up mmu context for new paging mode
  - look up new shadow roots
  - synchronize newly reachable shadow pages

Host translation updates:

  - mmu notifier called with updated hva
  - look up affected sptes through reverse map
  - drop (or update) translations

Further reading
===============

- NPT presentation from KVM Forum 2008
  http://www.linux-kvm.org/wiki/images/c/c8/KvmForum2008%24kdf2008_21.pdf