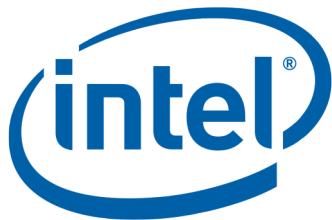


Shadow2

Xen Technical Summit, Summer 2006

Guilty parties:

**Tim Deegan (XenSource) &
Michael Fetterman (U of Cambridge, Intel)**



**UNIVERSITY OF
CAMBRIDGE**

Computer Laboratory

Shadow2

- **Full replacement for the old (shadow1) code.**
- **As of about two weeks ago, now just “shadow” in the xen-unstable tree.**



Some terminology:

- **2-level page tables == 32-bit page tables**
- **3-level page tables == 32-bit PAE page tables**
- **4-level page tables == 64-bit page tables**
- **0-level page tables == HVM guest with paging disabled**

- **M-on-N == guest with M-level page tables, running on Xen with N-level page tables**



Design Goals

- **PV and HVM guests**
- **2, 3, and 4 level Xen hypervisors**
- **2, 3, and 4 level PV guests (M-on-M)**
- **0, 2, 3, and 4 level HVM guests (M-on-N, $M \leq N$)**
- **Log dirty mode to support migration**
 - Want a fast way to write protect all pages
- **Translated PV guests (simplified MMU interface for guest)**
- **PV drivers for HVM domains (via grant tables)**
- **Superpages**
 - Superpages are currently splintered in the shadow
- **SMP guests**
 - Various different paging modes in each cpu simultaneously (during boot).
- **Better support for various research interests**
 - per-vcpu shadows
 - SMP precise replay
 - immutable memory
 - emulation-on-demand (aka V->E->V)
 - copy-on-write shadows



Initial focus

- **Correctness first, performance later**
- **Windows under HVM on PAE & 64-bit Xen**
- **Log dirty mode for PV guest migration**
- **Translated PV guests**



Major design choices

- **Emulate all writes to guest page tables**
- **Guest pages are shadowed separately for each page table type**
- **Single contiguous shadow for each guest page**
- **Allocate shadow memory from a fixed & dedicated pool**
- **Reverted P2M table back to page table format**
- **Start-of-domain shadow modes**
- **No reverse maps**



Emulate all writes to page tables

- **Shadow1 allowed guests to write to their page tables**
 - Kept track of out-of-sync guest page tables
 - Resync'ed on the next TLB sync op, and in various hypercalls
- **Shadow2 attempts to emulate all instructions which write to a guest page table**
 - Keeps Xen's shadow always in sync
 - If emulation fails for some reason, then remove all shadows of the page before allowing the update
- We left open the possibility of adding out-of-sync support back in later, but do not currently expect to add it



Emulate all writes: the demand fault case

- **Shadow1**

- Fault 1: reflected to the guest kernel, which then tries to update a PTE, causing...
- Fault 2: to take the page table page out of sync, xen copied the page first, and then shadowed it: 12KB touched.
When the guest retries the original faulting access, we get...
- Fault 3: Xen had to resync the now out-of-sync page: 12KB touched again.
- Various heuristics successfully reduced the amount touched

- **Shadow2**

- Fault 1: same as above
- Fault 2: The write is trapped and emulated, both guest and shadow page tables are updated
- There is no third fault, and the amount of cache thrash is greatly reduced



One shadow for each page table type

- **Types include:**

 - 32-bit level 2 table

 - 32-bit level 1 table

 - PAE level 3 table

 - PAE level 2 table

 - PAE level 1 table

 - 64-bit level 4 table

 - 64-bit level 3 table

 - 64-bit level 2 table

 - 64-bit level 1 table

- **A guest page may have one shadow for each of these types, or any subset, simultaneously.**

 - Handling linear page tables already required most of this
 - Writes to a guest page are reflected into all existent shadows simultaneously
 - Common case of a single type is unaffected for most operations

- **Makes handling SMP guests @ boot very simple...**



Size-mismatched PTEs

- **In the case of size mismatched page table entries (2-on-3, 2-on-4), a single guest page is shadowed by a contiguous set of shadows**
- **A 32-bit L1 guest page gets two contiguous shadow pages (4MB vs 2MB)**
- **A 32-bit L2 guest page gets four contiguous shadow pages (4GB vs 1GB)**
- **This allows a single shadow mfn to represent the entire shadow**



Shadow memory pool

- **Allocate shadow memory from a fixed & dedicated pool**
- **Recycle in a roughly LRU fashion when we run out**
- **Currently each domain has its own private pool**
- **New shadow_op hypercalls can resize the pool dynamically**



p2m in page table format

- **Simplifies the shadowing of HVM guests running in non-paging mode**
 - Same as a PV guest which is using the p2m table as its page table
- **IOMMUs may be able to use the same p2m table**



Start-of-domain shadow modes

- **Certain shadow modes can now only be enabled at domain creation time**
 - external vs internal
 - translated vs non-translated
 - shadow ref counts vs guest ref counts
- **Only two modes can now be enabled and disabled on-the-fly**
 - shadow test mode (only makes sense for PV domains)
 - log dirty mode (used for migration)



No reverse maps

- **The problem: shoot down**
 - Remove all write permissions for a given guest page
 - Remove all shadow mappings of a given guest page
- **We believe a couple simple heuristics will generally allow us to find the shadow entries pointing to that page**
- **Saves memory**
- **We think it can be better performance**
- **The APIs in the code are such that a reverse map can be easily added**
 - Various research projects want reverse maps



backup...



Interface to the rest of Xen

- **Changing a shadowed guest's CR3: write `v->arch.guest_table`, call `update_cr3()`**
 - `update_cr3()` updates `v->arch.cr3`
 - for PV domains: `write_ptbase()` always uses `v->arch.cr3`
 - for HVM domains: copy `v->arch.cr3` into `v->hvm.hw_cr3...`
- **Changing a shadowed guest's page table entries**
 - Take the `shadow_lock()`
 - Write the guest's page table entries
 - Call `shadow_validate_guest_entry()` or `shadow_validate_guest_pt_write()` to update the shadow tables
 - `shadow_unlock()`
- **Changing anything about a shadowed guest's paging mode**
 - Call `shadow_update_paging_modes()`



future optimization 1

- **Typically, when a page is promoted to a pagetable, it generally has exactly one writable mapping.**
- **And typically, when that mapping was originally installed, the writable count went from zero to one.**
- **We plan to keep a pointer in the page_info struct to the last writable mapping which caused each page's writable count to go from zero to one.**



future optimization 2

- **Fast-pathing some faults**
 - Excellent idea from the Virtual Iron shadow code
- **By storing a copy of the guest PTE's present & writable flags in two of the spare bits in the shadow PTE, we can fast-path certain operations, especially propagating a fault to the guest, without needing to even consult the guest pagetables.**
- **By putting a page's MMIO status in the third spare bit, we can also fast-path faults to MMIO space.**



future optimization 3 - batch updates

- **Detect bulk page table updates via either PV hints and/or simple heuristic.**
- **Unshadowing the guest page entirely.**
 - In the future, we can explore whether an "out of sync" mechanism would speed things up.
- **fork() as a special case:**
 - Unshadow the entire user portion of the guest address space, to save having to detect a "batch update" and unshadow each guest pagetable individually.



future optimization 4

- **The idea: read protect guest page tables**
- **While the page is unreadable by the guest, A&D bit updates need not be propagated from the shadow, and so we don't need to take A&D bit shadow faults at all.**
- **Either emulate reads to the guest page tables, or propagate A&D bits back, in bulk, on demand.**



future optimization 5

- **Teardown heuristics, specific to each guest OS**
- **For HVM and translated PV guests, prevents unnecessary faults when guest page tables get recycled as data pages**



multiple compilation

- **Much of the shadow code is compiled multiple times, once for each M-on-N combination.**
 - `arch/x86/mm/shadow/multi.c`
 - `arch/x86/mm/shadow/{multi,private,types}.h`
- **We are still thinking about compiling the PV vs HVM cases separately, too.**
- **For m-on-n mode, functions of the form**
`sh_[function_name]()`
are #define'd to
`sh_[function_name]__shadow_[n]_guest_[m]`
- **At the end of multi.c is a structure containing function pointers for each of the mode-specific functions; this is called shadow_entry (and gets the __shadow_[m]_guest_[n] suffix).**
 - `shadow_update_paging_modes()` updates `v->arch.shadow.mode`
- **To call the appropriate function, one generally calls**
`shadow_[function_name](v, [args])`
which is generally implemented by the following template:

```
[rettype] shadow_[function_name](v, [args]) {  
    return v->arch.shadow.mode->[function_name](v, [args]);  
}
```

