Paravirtualization in Xen

Xen and Paravirtualization

- Xen: most popular example of paravirtualized VMM
- Paravirtualization: modify guest OS to be amenable to virtualization
  - XenoLinux is a modified Linux OS that runs on Xen hypervisor
  - Application interface need not change
- Benefits: better performance than binary translation
- Disadvantages: requires source code changes to OS, porting effort
  - 1-2% code changes reported in the original Xen paper

“Xen and the Art of Virtualization”, Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield

Xen Architecture

- Type 1 hypervisor: runs directly over hardware
- Trap-and-emulate architecture
  - Xen runs in ring 0, guest OS in ring 1
  - Xen sits in the top 64MB of address space of guests
  - Guest OS traps to Xen to perform privileged actions
- A guest VM is called a domain
  - Special domain called dom0 runs control/management software

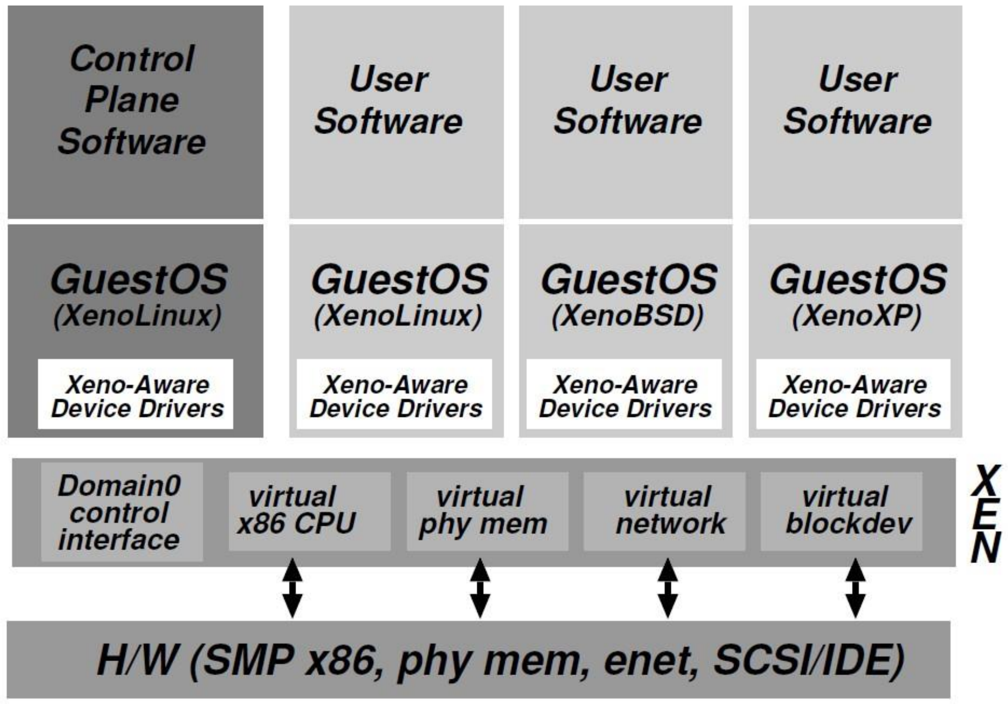


Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including *Domain0* running control software in a XenoLinux environment.

CPU virtualization in Xen

- Guest OS code modified to not invoke any privileged instruction
  - Any privileged operation traps to Xen in ring 0
- Hypercalls: guest OS voluntarily invokes Xen to perform privileged ops
  - Much like system calls from user process to kernel
  - Synchronous: guest pauses while Xen services the hypercall
- Asynchronous event mechanism: communication from Xen to domain
  - Much like interrupts from hardware to kernel
  - Used to deliver hardware interrupts and other notifications to domain
  - Domain registers event handler callback functions

Trap handling in Xen

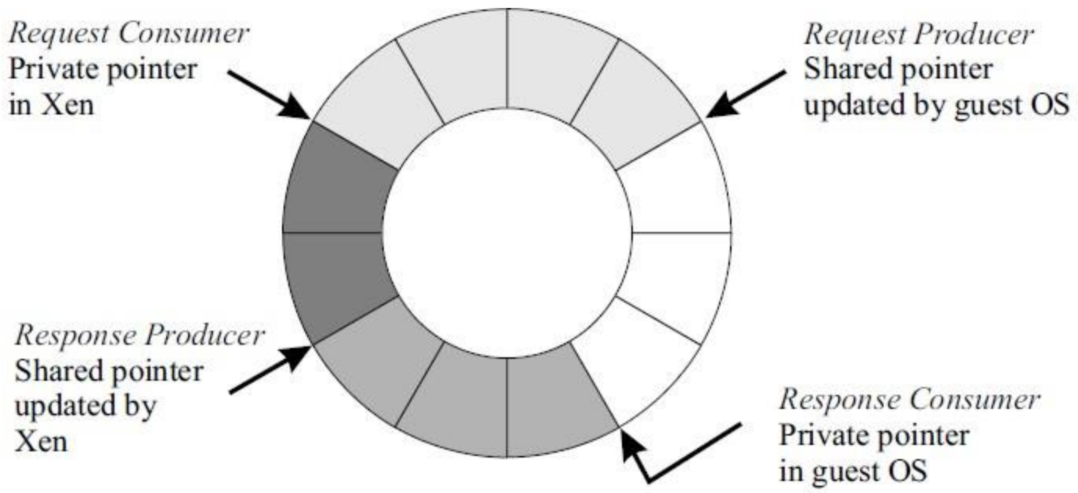
- When trap/interrupt occurs, Xen copies the trap frame onto the guest OS kernel stack, invokes guest interrupt handler
- Guest registers an interrupt descriptor table with Xen to handle traps
  - Interrupt handlers validated by Xen (check that no privileged segments loaded)
- Guest trap handlers work off information on kernel stack, no modifications needed to guest OS code
  - Except page fault handler, which needs to read CR2 register to find faulting address (privileged operation)
  - Page fault handler modified to read faulting address from kernel stack (address placed on stack by Xen)
- What if interrupt handler still invokes privileged operations?
  - Traps to Xen again and Xen detects this “double fault” (trap followed by another trap from interrupt handler code) and terminates misbehaving guest

Memory virtualization in Xen

- One copy of combined GVA→HPA page table maintained by guest OS
  - CR3 points to this page table
  - Like shadow page tables, but in guest memory, not in VMM
- Guest is given read-only access to guest “RAM” mappings (GPA→HPA)
  - Using this, guest can construct combined GVA→GPA mapping
- Guest page table is in guest memory, but validated by Xen
  - Guest marks its page table pages as read-only, cannot modify
  - When guest needs to update, it makes a hypercall to Xen to update page table
  - Xen validates updates (is guest accessing its slice of RAM?) and applies them
  - Batched updates for better performance
- Segment descriptor tables are also maintained similarly
  - Read-only copy in guest memory, updates validated and applied by Xen
  - Segments truncated to exclude top 64MB occupied by Xen

I/O virtualization in Xen

- Shared memory “rings” between guest domain and Xen/domain0
  - Front-end device driver in guest domain and backend in dom0
  - I/O requests placed in shared queue by guest domain
  - Request handled by Xen/domain0, responses placed in ring
  - Descriptors in queue: pointers to request data (DMA buffers with data for writes, empty DMA buffers for reads, etc.)
- Similar design to virtio
  - No copying of request data
  - Memory pages swapped between domains to exchange requests/responses
  - Batching for high performance



░ **Request queue** - Descriptors queued by the VM but not yet accepted by Xen

▓ **Outstanding descriptors** - Descriptor slots awaiting a response from Xen

▒ **Response queue** - Descriptors returned by Xen in response to serviced requests

□ **Unused descriptors**

Figure 2: The structure of asynchronous I/O rings, which are used for data transfer between Xen and guest OSes.

Summary

- Xen: paravirtualization based hypervisor
- Guest OS modified to suit virtualization
- Trap-and-emulate via VMM
  - Guest in ring 1, VMM in ring 0
  - Guest traps to VMM for privileged operations
- Combined GVA→HPA page tables in guest memory
  - Read-only copy in guest
  - Updated via hypercalls to Xen
- I/O via shared rings between guest and Xen/domain0