Structure of this Document

Outline

As a complex mature component of systems software, the Xen hypervisor has many APIs and ABIs.
The majority of these are poorly described, and some are undocumented entirely. There are multiple longstanding issues, causing problems ranging from packaging issues in distros, to the inability to run encrypted VMs. This set of documents is concerned with all facilities Xen provides to guests, and how guest kernels mediate access to these facilities for userspace.

This collection of documents are structured as follows:

- An enumeration of the current APIs and ABIs of Xen

- Challenges and limitations experienced with Xen's current interfaces, with explanations of the consequences and unwanted effects of the problems

- Requirements for improvement: solution-neutral statements of changes that need to be made
    - Establishing the scope for revision
    - Context important for determining acceptance criteria

- Change proposals
    - Taking into account: the current state of the technology; any requirements for providing support across a transition; available and expected capacity for design, development, review and deployment: what changes are proposed to be made?

Christopher Clark & Andrew Cooper
September 2025

Current APIs and ABIs

Xen APIs and ABIs

To understand the current design and interfaces of Xen's ABIs and APIs, an awareness of the history of Xen's development is informative.

Approximately the first half of this document will aim to provide a history and development background for the hypervisor, to provide context and explanation for design and decisions and a frame of reference for retrospective evaluation to assist design work for the future. The second half of this document will describe how Xen looks now: this is the material being evaluated for improvement.

Origins of Xen Interfaces and Guest Architectures

Development of the Xen hypervisor began with a design and implementation for paravirtualization of 32-bit x86 architecture machines.

32-bit Para-Virtualized on x86

Mechanically, PV guests under Xen are just like userspace under a kernel, so the API includes partition of the full virtual address space into separate regions between hypervisor, guest OS kernel and guest OS userspace. Guest kernels are necessarily aware of and exposed to the machine physical memory addresses used by the hardware, which may be discontiguous and fragmented, since guest kernels maintain their own active page tables under Xen's supervision. They are also provided with a contiguous physical address space abstraction, with guest physical frame numbers and a mapping to the corresponding hardware machine frame numbers, as support for the conventional contiguous memory model provided by hardware that the kernel is typically developed for.

In Xen's paravirtualized memory model for 32-bit x86, hypervisor control over the segmentation registers enables its protection from the guest OS kernel whilst remaining resident within the guest address space for efficient hypercall execution.

Hypervisor operations are invoked by the guest kernel either via hypercalls - ie. privileged software interrupts - or implicitly via traps or exceptions.

Early versions of Xen prior to 3.0 included hardware device drivers within the hypervisor kernel itself. In Xen 3.0, the architecture changed to move the hardware drivers into the privileged VM kernel of the first VM constructed at boot, Domain 0. This necessarily entailed development of the privileged Domain 0 interface. Work on XenStore and the grant and event interfaces enabled the split device driver model for implementing virtual network and storage devices for unprivileged virtual machines.

Xen on Itanium (2005)

Support for ia64 architecture; later deprecated and removed but this hardware was an architectural consideration throughout the first years of Xen development.

32-bit PAE on x86 (2005)

Support for larger physical address space with Physical Address Extensions was developed fairly early in Xen for x86 (2005), adding support for 3-level page tables both within the hypervisor and guest VMs. 32-bit PAE remains a separate ABI.

64-bit PV on x86 (2005)

Since 64-bit mode on x86-64 did not support segmentation when introduced, disjoint page tables are maintained for execution of guest kernel mode and guest user mode. This structure has more recently ensured robust enforcement of isolation between the system privilege levels as mitigation against Meltdown.

32-bit and 64-bit Hardware Virtual Machines on x86 (2006)

Developed for Intel VTx and AMD-V hardware features, and including support for PAE. This execution mode was required to provide support for running unmodified guest OS kernels, including Windows XP: in this mode, the hypervisor is not resident within the guest OS address space and the guest maintains its own page tables that contain fully virtualized guest physical frame numbers. It is supported by a privileged device model process that runs in an entirely separate address space for system device emulation.

With hardware support for multi-level page table translation, the guest page tables are translated to machine frame numbers via a second set of tables maintained by the hypervisor. Prior to availability of efficient hardware support for this second translation, a separate technique was implemented with the hypervisor maintaining shadow page tables for each guest, performing translation of guest accesses to their writable page tables via emulation. This enables the hardware to run on the shadow tables that contain machine frame numbers and so avoid the need for a second translation by hardware. Shadow page tables are also useful for enabling guest introspection, which was developed later.

Simultaneous Multi Processing

Multiprocessor support followed after the initial development of support for guest long mode.

Stub domains and System Disaggregation

Support for running the privileged device model process in a separate virtual machine context provides hardware-based isolation and confinement to the emulator software, providing protection against the effects of defects in the emulator or attacks upon it by malicious software.

Further system disaggregation is enabled by allowing the hypervisor toolstack to run in a separate (control) domain from the hardware device drivers, and allowing PCI devices to be dedicated to separate virtual machines that mediate access to those specific devices on behalf of other virtual machines.

Managing permissions for privileged operations by VMs is possible either by using the hypervisor default internal security policy, which will typically allow privileged operations by

Domain 0 and generally not by other domains, with some limited exceptions for device models or passthrough devices managed by the toolstack, or via a dedicated permission enforcement subsystem that takes an administrator-provided system security policy file (XSM/Flask).

The ability to securely distribute functions and logical roles to different virtual machines, adhering to the principles of least privilege to support maintaining system security, necessarily introduces some complexity in the design of hypercall operations that perform the individual implementation steps of the larger composite procedures.

Xen on ARM

Support for Arm architecture developed following the x86 HVM-mode interface, without requiring the device model for emulation. 64-bit support was developed after the initial 32-bit system support.

PVH on x86

A new hybrid mode of execution has been developed for enlightened Xen-aware paravirtualized guest OSes, such as Linux and FreeBSD: able to use the silicon-assisted virtualization support for the HVM fully-isolated system memory model, but without the need to perform system initialization according to legacy standards and the requirement for the device model emulator.

Xen on PPC

Previously developed, deprecated and then more recently has seen new interest in resuming development.

Xen on RISC-V

Under development, with the design drawing significantly upon the Xen-on-Arm architecture.

Xen interfaces, APIs and ABIs

Note: this interface list is non-exhaustive. Not all items are pertinent to the Xen hypercall ABI design.

- Hypervisor Entry
    - Entry: multiboot 1
    - Entry: multiboot 2
    - Entry: EFI
        - Secure Boot measurement
    - Entry: PVH
        - kexec
    - Hypervisor command line
        - Graceful failure handling for misconfigurations
    - Hyperlaunch + Dom0less
    - Processor microcode application
    - XSM/Flask policy loading
    - tboot log access
    - Resume from host S3 / S0ix
    - Watchdog, crash kernel
- Guest Entry
    - Start info page
    - Shared info page
    - Initial vCPU registers content from domain_create
        - contrast vs. default state of a physical CPU
    - Initially-populated memory
    - Initially-populated grant table
    - Console IO
    - CPUID: Available processor feature bits
    - Model Specific Registers
    - ACPI tables
    - DMI tables
    - Guest BIOS
    - Memory map, address ranges
    - Processor topology
    - Processor frequency regulation
    - Memory locality
    - Timers
    - Guest power states
    - Guest kernel ABI declaration: Xen-x86-64-3.0
    - CPU exception handling
- Hypercall interface
    - PV32
    - PV32pae
    - PV64
    - HVM32
    - HVM64
    - Compat

    - Register conventions
    - Use of Virtual Addresses: PV vs HVM/PVH
    - Reference example with bounds checking: DMOP
    - GHCB: communications paths for registers and hypercall issue, marshalling protocol
- Guest syscalls
    - Flag cleared: syscall behaves differently to any other trap in a PV guest
- XenBus
- XenStore
- Event Channels
- Grants
- PV devices
    - Storage
    - Network
    - Console
    - Timers
- Privcmd
- Argo
    - Access to specific rings
    - Connectivity between domains of different classes
- Foreign Mapping
- MMIO emulation
- IOReq Server interfaces
- Emulated devices
- Guest framebuffer
- Guest sound devices
- Guest optical media devices
- Guest USB 2.0
- Viridian enlightenments
- Xen PCI device
- PCI passthrough
- Stubdomains
- Xen Toolstack interfaces
    - Tools interface to invoke toolstack operations
    - Multi-host toolstack interfaces
    - Host storage interfaces
    - Host network interfaces
- Domain builder
- Shim: PV guest execution within a PVH domain
- Live Migration
- Live Patching
- Introspection
- Memory Ballooning
- CPU Hotplug
- PCI device hotplug
- Guest monitor display hotplug
- Crash kernel
- VirtIO guest devices

A list of hypercalls to be added, with annotation for each, indicating:

- Privileged or not: is this hypercall operating on domain-wide resources?
    - as such should typically be constrained to access only by a guest kernel, rather than unprivileged user-space
- Logical operation or not: for example: this is performing a standard part of domain construction on behalf of another domain, vs. this a narrowly-defined, mechanistic operation
    - logical operations could be expected to be performed by a userspace process (toolstack)
- Fast-path vs. Slow-path operations
    - noted with reference to Hyper-V's different register-passing interface that is used for fast-path hypercalls
    - to support prioritizing and enabling the fast execution of operations that are important for performance

Current Limitations

Current Limitations in Xen ABIs and APIs

Brief: Challenges and limitations experienced with Xen's current interfaces, with explanations of the consequences and unwanted effects of the problems.

To be added: cross-references to specific identified items in the Current APIs and ABIs document.

Limitations encountered and improvements wanted

Improved support for discovery, enumeration of interfaces, capabilities

A general point applicable to the design of many of Xen's existing interfaces.
Without enumeration, interfaces are expected to exist and are not configurable.

Versioning of interfaces

The current unstable interface to privileged domains from the hypervisor has resulted in having to recompile the kernels and device models within privileged domains upon hypervisor version upgrade.

Toolstack control over guest exposure to and access to individual interfaces

A general requirement for new interfaces being introduced. This should include support for audit of guest access to privileged interfaces.

Support for running encrypted VMs

Hypervisor cannot access guest memory without prior guest authorization. Precludes general use of guest virtual addresses in hypervisor interfaces: accesses must be page-based, with reference within the guest physical page space, in pages that have been identified as shared with the hypervisor.
The same interface will also be usable within unencrypted VMs.

Use of Virtual Addresses in the Xen HVM hypercall ABI

Use of guest virtual addresses in the hypercall interface is correct for the x86 PV guest ABIs, but not for the others.
Guest virtual addresses require translation through two sets of page tables: guest virtual to guest physical, and guest physical to machine. This translation is expensive - ie. can and should be reduced with a different interface - and is also disallowed for encrypted VMs.

Virtual addresses are acceptable for some PV guest interfaces since the continuation of their use entails fewer changes required for existing PV guests.

Mapping guest resources by virtual address

The shared info and timestruct pages are mapped by virtual address for the hypervisor to write into: these should be deprecated as legacy interfaces and replaced with an interface that requires physical addresses instead.

Hypercall interfaces: review and standardization

Revise design of interfaces according to current known best-practices: methods for passing memory descriptors, efficient use of registers for arguments, dmop-style arguments for userspace-invoked operations, standardized interface across varying guest types, etc.

Standardizing error responses

Review of existing interfaces for opportunities to standardize error response codes across similar operations

Design for accommodation of differing page sizes

Common to see shift-by-12 in logic operations on x86 with expectation of 4K, but some Arm guest kernels may prefer 16K and address-based interface design may support more useful abstractions.
x86 PV guests are currently disallowed access to using superpages.

Arm ABI: resolution of 32/64 bit register state and 128 bit architecture

Current ABI: all pointers are uint64_t, so in-memory structures are the same on arm32 and arm64.
To be resolved: issue of differing register state of arm32 vs arm64, and design for arrival of 128 bit architecture.
For ABI design: key point is the: guest kernel idea of address space size.
ie. The guest kernel bitness is significant. Hypervisor to perform sign extension or zero extension as appropriate: this provides a cleaner design, more appropriate for future development.

Shared info page: layout varies per guest 32/64 bitness

Unhelpful for SMP domains. How to transition to a new fixed layout?

Privcmd

Presents challenges for system administration; a new design for an alternative interface is requested.

Domain create operation complexity

The current design of this interface deserves revisiting.
eg. Mechanism for nomination and communication of the domain identifier (x86) or interrupt controller (Arm) upon creation complicates the interface from what could otherwise be input-only, which further complicates the logic between the kernel and userspace.

For consideration: invocation of sysctls or domctls to perform domain configuration after the initial allocation completes but prior to scheduling.

Timer provision to PV guests

Default 100Hz timer configured: legacy from early design aiming to optimize in support of the guest kernel but resulting in an interface that differs from behaviour expected of physical hardware.

Dynamic memory allocation, ballooning

Present interface is an operation between a guest and the hypervisor: enables severe fragmentation across the guest physical address space, which can affect performance of operations involving the p2m map due to that.
Should instead be an operation between the guest and the toolstack as a logical operation, and for the toolstack to manage instead. The hypervisor needs to implement a rate limiter on expensive operations invoked by the guest or on its behalf.

Legacy architectural optimization

eg. TS flag clearing on syscall entry

Conflation of privileged and logical operations

Some hypervisor primitives perform operations that are not optimally abstracted, where the logical operation being requested is conflated with the privileged implementation required to perform it.
eg. physdev op
eg. mmuext op
eg. toolstack invocation of event_channel_alloc_unbound for a target domain
Consider separation into separate calls for a kernel requesting operation upon itself, and a domain builder acting on behalf of a third party domain under construction.

Secure boot: support for non-monolithic kernel systems

Design for support of systems with separate Hardware Domain and Control Domain, where userspace for either may be considered outside the TCB for a particular system.
XSM/Flask can express constraints eg. on the ability of the control domain to overwrite the hardware domain to enable reflexive access back for overwrites within the control domain.

Host UEFI Secure Boot

Downstream distributions of Xen currently implement host UEFI Secure Boot with some caveats and modifications to Xen and system boot components.

Security Support for XSM/Flask

(A list of work items to be produced towards enabling security-support for XSM/Flask upstream)

Deprecation of source code headers for interface definition

An objective is to transition from the current C headers being the canonical source of correctness for the ABI to a clear descriptive and authoritative statement with a corresponding C structure that implements it. The interface can be stated in a language-neutral way without ambiguity, to bring clarity around issues of padding, sign extension, internal alignment, differences on 64 vs 32 bit, etc.
This work can support development of an IDL at a later time.

Support for Linux distro device models and hypervisor upgrades

QEMU typically runs within a privileged domain on Xen systems, to provide device emulation or access to storage, and the ABI from the Xen hypervisor to privileged domains is unstable, changing with each Xen major release. This has precluded the distribution of QEMU binaries by Linux distros that can support arbitrary versions of the hypervisor, since the hypervisor to device model interface is not well defined. It also prevents upgrade of the hypervisor beneath an existing system running the toolstack.
This should be addressed with explicit interface design for both forwards and backwards version compatibility between hypervisor and toolstack, and processes for ensuring that retaining the stability of interface operations can be achieved.
This will significantly improve the deployability of the Xen hypervisor and support for tooling and software systems that integrate with Xen.

Resource contention on access

Hypercalls or domain operations may require access to central resources where contention is either predictable or costly. Design documentation should describe approaches for mitigating this and interface design should support these.
To evaluate for reference: Linux netlink interface.

Principles for improvements

Principles for Improvement and Future Development

*Brief: Requirements for improvement: **solution-neutral statements** of changes that need to be made*

- *Establishing the scope for revision*
- *Context important for determining acceptance criteria of change proposals*

This document will be expanded upon as feedback is incorporated from the preceding two documents.

General Principles

Principle of least surprise

Design with reference to the expected behaviour of physical hardware, and foreseeable evolution of hardware over time, when determining default behaviours for interfaces.
eg. determining appropriateness for automatic flag clearing on entry, vs. potential optimization.
Examine available methods to enable guests to explicitly opt-in when implementing divergent behaviour.

Design with reference to distinguishing privileged guest operations

Identify which guest operations are privileged, eg. operating upon domain-wide resources such as grant-table entries, and so ought be constrained to use by privileged software such as a guest OS kernel, versus non-privileged or logical operations, and thereby appropriate for access by guest userspace software. This can enable efficient and appropriate audit and access control enforcement.

Versioning, deprecation and availability of prior hypervisor interfaces

Since encrypted VMs cannot invoke or access the current hypercall interface, there is no need to preserve their ability to access it: they can be assumed to be compatible with the first version of the newer interface.

In general, guest VMs that are non-control domain, non-hardware domain, must be supported as entirely backwards compatible. The expectation for all hypercalls is that they are stable.

Reference to adjacent hypervisor design

- The Hyper-V Hypervisor Top Level Functional Specification is a reasonable document.
  All hypercalls are performed in registers, addresses are guest physical with metadata in the low address bits.

    - Useful for reference to Xen control plane operations.
- The Hyper-V Primary Partition is similar to Xen’s Domain-0, with the rest of the architecture similar to KVM.
- KVM domain construction is markedly different than in Xen, so many privileged operations are omitted., eg. domctls, sysctls
- The uXen hypervisor has design heritage with Xen with subsequent development occurring in parallel, so provides some references useful for comparison.
- pKVM has a system architecture of interest.

Specific Improvements to be Made

Use of Physical Addresses in HVM-mode hypervisor interfaces

- use a byte-addressable physical address instead of a page reference
- address rather than page references appropriate for use on systems using multiple page sizes
- allows use of the low bits for flags, similar to how MSRs do

All data produced and consumed by the hypervisor shall be expressed as guest physical addresses.

Distinguishing fast-path from slow-path operations

Explicitly identifying operations that are performance-critical will help ensure that there is consensus on the need to implement eg. avoidance of lock contention, at the potential cost of higher complexity within the hypervisor.

New APIs and ABIs