

NOVA Microhypervisor Interface Specification

Udo Steinberg
udo@hypervisor.org

February 13, 2013

Preliminary

Preliminary

**Copyright © 2006–2011 Udo Steinberg, Technische Universität Dresden.
Copyright © 2012–2013 Udo Steinberg, Intel Corporation.**

This specification is provided "as is" and may contain defects or deficiencies which cannot or will not be corrected. The author makes no representations or warranties, either expressed or implied, including but not limited to, warranties of merchantability, fitness for a particular purpose, or non-infringement that the contents of the specification are suitable for any purpose or that any practice or implementation of such contents will not infringe any third party patents, copyrights, trade secrets or other rights.

The specification could include technical inaccuracies or typographical errors. Additions and changes are periodically made to the information therein; these will be incorporated into new versions of the specification, if any.

Contents

| | | |
|------------|--|-----------|
| I | Introduction | 1 |
| 1 | System Architecture | 2 |
| II | Basic Abstractions | 3 |
| 2 | Kernel Objects | 4 |
| 2.1 | Protection Domain | 4 |
| 2.2 | Execution Context | 4 |
| 2.3 | Scheduling Context | 5 |
| 2.4 | Portal | 5 |
| 2.5 | Semaphore | 5 |
| 3 | Mechanisms | 6 |
| 3.1 | Scheduling | 6 |
| 3.2 | Communication | 6 |
| 3.3 | Exceptions and Intercepts | 7 |
| 3.4 | Interrupts | 7 |
| 3.5 | Capability Delegation | 7 |
| 3.6 | Capability Revocation | 8 |
| 3.7 | Device Assignment | 8 |
| III | Application Programming Interface | 10 |
| 4 | Data Types | 11 |
| 4.1 | Capability | 11 |
| 4.1.1 | Null Capability | 11 |
| 4.1.2 | Memory Capability | 11 |
| 4.1.3 | Port I/O Capability | 11 |
| 4.1.4 | Object Capability | 11 |
| 4.1.5 | Reply Capability | 13 |
| 4.2 | Capability Selector | 13 |
| 4.3 | Capability Range Descriptor | 14 |
| 4.3.1 | Null Capability Range Descriptor | 14 |
| 4.3.2 | Memory Capability Range Descriptor | 14 |
| 4.3.3 | Port I/O Capability Range Descriptor | 14 |
| 4.3.4 | Object Capability Range Descriptor | 14 |
| 4.4 | Message Transfer Descriptor | 15 |
| 4.5 | Quantum Priority Descriptor | 15 |
| 4.6 | User Thread Control Block | 16 |
| 4.6.1 | Header Area | 16 |
| 4.6.2 | Data Area | 16 |

| | | |
|-----------|---|-----------|
| 5 | Hypercalls | 19 |
| 5.1 | Definitions | 19 |
| 5.2 | Communication | 20 |
| 5.2.1 | Call | 20 |
| 5.2.2 | Reply | 21 |
| 5.3 | Capability Management | 22 |
| 5.3.1 | Create Protection Domain | 22 |
| 5.3.2 | Create Execution Context | 23 |
| 5.3.3 | Create Scheduling Context | 24 |
| 5.3.4 | Create Portal | 25 |
| 5.3.5 | Create Semaphore | 26 |
| 5.3.6 | Revoke Capability Range | 27 |
| 5.3.7 | Lookup Capability Range | 28 |
| 5.4 | Execution Control | 29 |
| 5.4.1 | Execution Context Control | 29 |
| 5.4.2 | Scheduling Context Control | 30 |
| 5.4.3 | Semaphore Control | 31 |
| 5.5 | Device Control | 32 |
| 5.5.1 | Assign PCI Device | 32 |
| 5.5.2 | Assign Global System Interrupt | 33 |
| 6 | Booting | 34 |
| 6.1 | Root Protection Domain | 34 |
| 6.1.1 | Resource Access | 34 |
| 6.1.2 | Initial Configuration | 34 |
| 6.2 | Hypervisor Information Page | 35 |
| IV | Application Binary Interface | 38 |
| 7 | ABI x86-32 | 39 |
| 7.1 | Initial State | 39 |
| 7.2 | Event-Specific Capability Selectors | 39 |
| 7.3 | UTCB Data Area Layout | 42 |
| 7.4 | Message Transfer Descriptor | 44 |
| 7.5 | Calling Convention | 45 |
| 8 | ABI x86-64 | 49 |
| 8.1 | Initial State | 49 |
| 8.2 | Event-Specific Capability Selectors | 49 |
| 8.3 | UTCB Data Area Layout | 52 |
| 8.4 | Message Transfer Descriptor | 55 |
| 8.5 | Calling Convention | 56 |
| V | Appendix | 60 |
| A | Acronyms | 61 |
| B | Bibliography | 63 |
| C | Console | 64 |
| D | Download | 65 |

Part I

Introduction

Preliminary

1 System Architecture

The NOVA OS Virtualization Architecture facilitates the coexistence of multiple legacy guest operating systems and a multi-server user environment on a single platform [4]. The core system leverages virtualization technology provided by recent x86 platforms and comprises the Microhypervisor and one or more Virtual-Machine Monitors (VMMs).

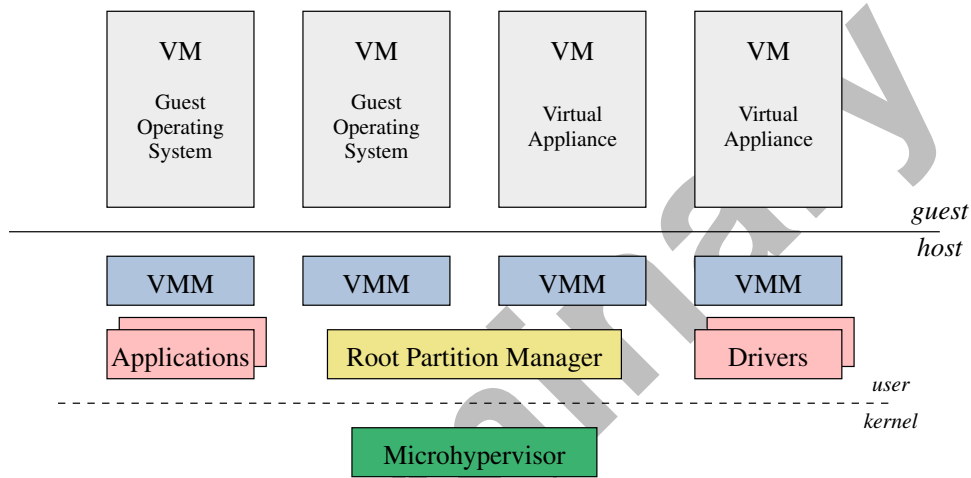


Figure 1.1: System Architecture

Figure 1.1 shows the structure of the system. The microhypervisor is the only component running in privileged root/kernel mode. It isolates the user-level servers, including the virtual-machine monitor, from one another by placing them in different address spaces in unprivileged root/user mode. Each legacy guest operating system runs in its own virtual-machine environment in non-root mode and is therefore isolated from the other components.

Besides isolation, the microhypervisor also provides mechanisms for partitioning and delegation of platform resources, such as CPU time, physical memory, I/O ports and hardware interrupts and for establishing communication paths between different protection domains.

The virtual-machine monitor handles virtualization faults and implements virtual devices that enable legacy guest operating systems to function in the same manner as they would on bare hardware. Providing this functionality outside the microhypervisor in the VMM considerably reduces the size of the trusted computing base for the multi-server user environment and for applications that do not require virtualization support.

The architecture and interfaces of the VMM and the multi-server user environment are not described in this document.

Part II

Basic Abstractions

Preliminary

2 Kernel Objects

2.1 Protection Domain

1. The **Protection Domain (PD)** is a unit of protection and isolation.
2. A protection domain is referenced by a **Protection Domain Capability ($CAP_{OBJ_{PD}}$)**.
3. A protection domain is composed of a set of spaces that hold capabilities to platform resources or kernel objects that can be accessed by execution contexts within the protection domain. The following spaces are currently defined:
 - Memory Space
 - Port I/O Space
 - Object Space
4. The memory space of a protection domain holds capabilities to page frames in physical memory.
5. The port I/O space of a protection domain holds capabilities to I/O ports.
6. The object space of a protection domain holds capabilities to the following kernel objects:
 - **Protection Domain (PD)**
 - **Execution Context (EC)**
 - **Scheduling Context (SC)**
 - **Portal (PT)**
 - **Semaphore (SM)**

2.2 Execution Context

1. The **Execution Context (EC)** is an abstraction for an activity within a protection domain.
2. An execution context is referenced by an **Execution Context Capability ($CAP_{OBJ_{EC}}$)**.
3. An execution context is permanently bound to the protection domain in which it was created.
4. An execution context may optional have a scheduling context bound to it.
5. There exist two flavors of execution context:
 - Kernel thread
 - Virtual CPU
6. An execution context comprises the following information:
 - Reference to protection domain (2.1)
 - Event Selector Base (**SEL_{EVT}**) (3.3)
 - Reply capability register (4.1)
 - **User Thread Control Block (UTCB)** (4.6)
 - Central Processing Unit (**CPU**) registers (architecture dependent)
 - Floating Point Unit (**FPU**) registers (architecture dependent)

2.3 Scheduling Context

1. The **Scheduling Context (SC)** is a unit of dispatching and prioritization.
2. A scheduling context is referenced by a **Scheduling Context Capability** ($CAP_{OBJ_{SC}}$).
3. A scheduling context is permanently bound to exactly one physical CPU.
4. At any point in time, a scheduling context is bound to exactly one execution context.
5. Donation of a scheduling context to another execution context binds the scheduling context to that other execution context.
6. A scheduling context comprises the following information:
 - Reference to execution context (2.2)
 - Time quantum
 - Priority

2.4 Portal

1. A **Portal (PT)** represents a dedicated entry point into the protection domain in which the portal was created.
2. A portal is referenced by a **Portal Capability** ($CAP_{OBJ_{PT}}$).
3. A portal is permanently bound to exactly one execution context.
4. A portal comprises the following information:
 - Reference to execution context (2.2)
 - **Message Transfer Descriptor (MTD)** (4.4)
 - Entry instruction pointer

2.5 Semaphore

1. A **Semaphore (SM)** provides a means to synchronize execution and interrupt delivery by selectively blocking and unblocking execution contexts.
2. A semaphore is referenced by a **Semaphore Capability** ($CAP_{OBJ_{SM}}$).

3 Mechanisms

3.1 Scheduling

The microhypervisor implements a round-robin scheduler with multiple priority levels. Whenever an execution context is ready to execute, the runqueue contains all scheduling contexts bound to that execution context. When an execution context blocks, the microhypervisor removes the corresponding scheduling contexts from the runqueue.

When the microhypervisor needs to make a scheduling decision, it selects the highest-priority scheduling context from the runqueue and dispatches the execution context bound to that scheduling context.

The parameters of a scheduling context influence the scheduling behavior of the system as follows:

- The priority defines the importance of a scheduling context. A higher-priority scheduling context always has precedence and immediately preempts a lower-priority scheduling context.
- The time quantum defines the number of microseconds that the execution context, which is currently bound to the scheduling context, can utilize the CPU when it is dispatched. A dispatched execution context consumes the time quantum of its scheduling context until the quantum reaches zero; at that point the microhypervisor preempts the execution context, replenishes the time quantum of the scheduling context, and makes a scheduling decision.

3.2 Communication

Message passing between protection domains is governed by portals. A portal represents a dedicated entry point into the protection domain to which the portal is bound. An execution context in a protection domain can call any portal for which the protection domain holds a capability. Portal capabilities can be delegated in order to establish cross-domain communication channels.

To initiate a message-passing operation from one protection domain to another, the caller execution context passes a portal capability selector $SEL_{OBJ_{PT}}$ to the microhypervisor. The microhypervisor uses the capability selector to look up the portal capability $CAP_{OBJ_{PT}}$ in the object space of the caller protection domain. If the lookup succeeds, the microhypervisor loads the destination protection domain and entry instruction pointer for that domain from the portal.

An arbitrary number of portals can be bound to a callee execution context in a protection domain. The callee provides the stack for handling one incoming request on any of these portals. If the callee is busy handling another request, and both caller and callee are on the same CPU, the caller may optionally lend its scheduling context to the callee to help it run the previous request to completion.

Once the callee is available to handle a new request and a caller exists for any portal bound to the callee, the microhypervisor arranges a rendezvous and transfers the message from the **UTCB** of the caller to the **UTCB** of the callee.

If the request established a reply capability for the callee, the callee may subsequently respond directly to the caller through a reply operation without risking to block, because the caller is already waiting for the response.

The following forms of message passing are currently supported:

Nondonating Call

During a nondonating call, the caller execution context traverses the destination portal, rendezvouses with a callee execution context and transfers a message to it. The microhypervisor establishes a reply capability for the callee. The caller blocks on the instruction following the hypercall and does *not* donate the current scheduling context to the callee. The callee may later invoke the reply capability to send a response directly to the blocked caller. Upon receiving the response the caller becomes unblocked.

Donating Call

A donating call differs from a nondonating call in that the caller donates the current scheduling context to the callee. The donation mechanism implements priority and bandwidth inheritance from the caller to the callee. The caller blocks on the instruction following the hypercall and the callee starts executing immediately. The microhypervisor also establishes a reply capability for the callee. The callee may later invoke the reply capability to send a response directly to the blocked caller. Upon receiving the response the caller becomes unblocked.

Reply

The reply operation sends a message back to the caller identified by the reply capability and revokes the reply capability. If the reply capability was established by a donating call, the microhypervisor returns the previously donated scheduling context back to the caller. The callee blocks until the next request arrives.

3.3 Exceptions and Intercepts

When an execution context triggers a hardware exception or VM intercept, the microhypervisor adds the exception number or intercept reason to the Event Selector Base (SEL_{EVT}) of the affected EC. If the resulting capability selector refers to a portal capability $CAP_{OBJ_{PT}}$, the microhypervisor arranges an implicit donating call for the execution context through the corresponding portal; otherwise the execution context is shut down.

The entire handling of the exception or intercept is performed using the current scheduling context of the execution context that triggered the event. Furthermore, that execution context remains blocked until the handler has replied with a message to resolve the exception or intercept.

The number of capability selectors used for exception and intercept handling is conveyed in the [Hypervisor Information Page \(HIP\)](#) (6.2). The translation of hardware exception numbers and intercept reasons to capability selectors is described in the processor-specific [Application Binary Interface \(ABI\)](#) (IV).

3.4 Interrupts

The microhypervisor provides a semaphore per Global System Interrupt (GSI) [2]. An execution context waits for an interrupt by performing an `sm_ctl[down]` hypercall to block on the corresponding semaphore. When the interrupt occurs, the microhypervisor issues an `sm_ctl[up]` operation for the semaphore.

3.5 Capability Delegation

Delegation of capabilities from one protection domain to another is performed during communication. The execution context that sends a message puts typed items in its [UTCB](#), specifying which range of capabilities

from the sender's protection domain it wants to delegate to the receiver's protection domain. The receiver specifies in its [UTCB](#), which range of capabilities it is willing to accept and where they should be installed in the receiver's protection domain.

The microhypervisor computes the intersection of the sender and receiver ranges and delegates only those capabilities that are covered by both ranges. The sender may optionally reduce the permissions of the delegated capabilities for the receiver, using the mask field in the [Capability Range Descriptor \(CRD\)](#).

If the capability ranges of the sender and receiver differ in size, the capability hotspot, specified by the sender, is used for disambiguation as illustrated in Figure 3.1.

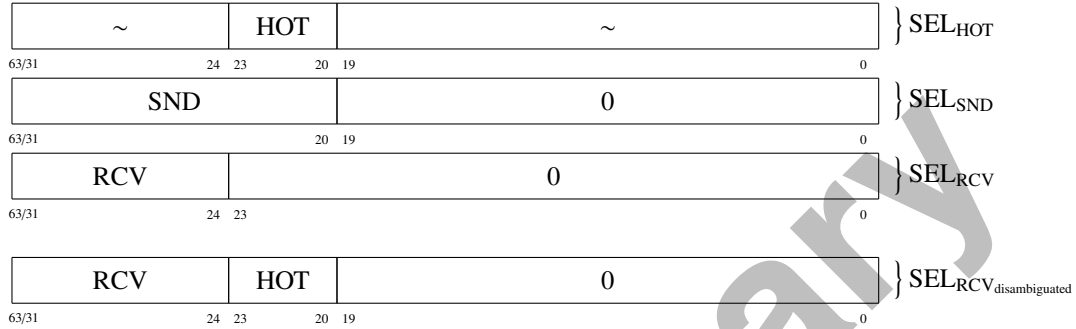


Figure 3.1: Capability Range Disambiguation

In this example, the sender has specified a capability range of order 20, starting at SEL_{SND} , whereas the receiver has specified a capability range of order 24, starting at SEL_{RCV} . There exist 2^4 possible locations in the receiver range, where the sender range could be delegated. Whenever two capability ranges differ in size, the microhypervisor truncates the larger range by taking the ambiguous bits from the capability hotspot.

3.6 Capability Revocation

[Capability](#) permissions may be revoked at any point in time. During the [revoke](#) hypercall, the execution context supplies a [Capability Range Descriptor \(CRD\)](#), whose mask field describes which permissions to revoke from all capabilities in the specified range. For each bit set in the mask, the microhypervisor removes the corresponding bit in the capability permissions.

Revoking permissions from a capability also revokes those permissions from all inherited capabilities in the same or other protection domains.

Once all permissions of a capability have been removed, the hypervisor deletes that capability. When all capabilities and references to a kernel object have been deleted, the hypervisor destroys the kernel object.

3.7 Device Assignment

The microhypervisor provides mechanisms for direct assignment of PCI devices to [VMs](#), and for implementing user-level device drivers safely. The component that manages the PCI device (e.g., VMM or device driver) must perform the following steps in any order:

- It must assign the device, via the [Assign PCI](#) hypercall, to the protection domain that implements the driver for the device.

- If the device performs [DMA](#), the protection domain to which the device is assigned must have mapped the respective memory regions as [DMA-enabled](#). This can be achieved by setting the D-bit in the [typed item](#) that establishes the memory region.
- If the device generates interrupts, the interrupt must be configured via the [Assign GSI](#) hypercall.

Preliminary

Part III

Application Programming Interface

4 Data Types

4.1 Capability

A **Capability** (CAP) is a reference to a kernel object plus associated auxiliary data, such as access permissions. Capabilities are opaque and immutable to the user — they cannot be inspected, modified or addressed directly; instead user programs access a capability via a capability selector (4.2). All capabilities can be delegated and revoked as described in Section 3.5. The following types of capabilities exist:

4.1.1 Null Capability

A **Null Capability** (CAP_0) does not reference anything and there are no permissions defined.

4.1.2 Memory Capability

A **Memory Capability** (CAP_{MEM}) references a 4KB page frame. It is stored in the memory space of a protection domain. The capability permissions are defined as follows:

| | | | | |
|---|---|---|---|---|
| ~ | ~ | x | w | r |
| 4 | 3 | 2 | 1 | 0 |

r readable if set.

w writable if set.

x executable if set.

4.1.3 Port I/O Capability

An **Port I/O Capability** (CAP_{PIO}) references an I/O port. It is stored in the port I/O space of a protection domain. The capability permissions are defined as follows:

| | | | | |
|---|---|---|---|---|
| ~ | ~ | ~ | ~ | a |
| 4 | 3 | 2 | 1 | 0 |

a accessible if set.

4.1.4 Object Capability

An **Object Capability** (CAP_{OBJ}) references a kernel object. It is stored in the object space of a protection domain. The following types of object capabilities are currently defined:

4.1.4.1 Protection Domain Capability

A [Protection Domain Capability](#) ($CAP_{OBJ_{PD}}$) references a protection domain (2.1). The capability permissions are defined as follows:

| | | | | |
|----|----|----|----|----|
| sm | pt | sc | ec | pd |
| 4 | 3 | 2 | 1 | 0 |

pd Hypercall `create_pd` (5.3.1) permitted if set.

ec Hypercall `create_ec` (5.3.2) permitted if set.

sc Hypercall `create_sc` (5.3.3) permitted if set.

pt Hypercall `create_pt` (5.3.4) permitted if set.

sm Hypercall `create_sm` (5.3.5) permitted if set.

4.1.4.2 Execution Context Capability

An [Execution Context Capability](#) ($CAP_{OBJ_{EC}}$) references an execution context (2.2). The capability permissions are defined as follows:

| | | | | |
|---|----|----|---|----|
| ~ | pt | sc | ~ | ct |
| 4 | 3 | 2 | 1 | 0 |

ct Hypercall `ec_ctrl` (5.4.1) permitted if set.

sc Hypercall `create_sc` (5.3.3) can bind a scheduling context if set.

pt Hypercall `create_pt` (5.3.4) can bind a portal if set.

4.1.4.3 Scheduling Context Capability

A [Scheduling Context Capability](#) ($CAP_{OBJ_{SC}}$) references a scheduling context (2.3). The capability permissions are defined as follows:

| | | | | |
|---|---|---|---|----|
| ~ | ~ | ~ | ~ | ct |
| 4 | 3 | 2 | 1 | 0 |

ct Hypercall `sc_ctrl` (5.4.2) permitted if set.

4.1.4.4 Portal Capability

A [Portal Capability](#) ($CAP_{OBJ_{PT}}$) references a portal (2.4). The capability permissions are defined as follows:

| | | | | |
|---|---|---|---|------|
| ~ | ~ | ~ | ~ | call |
| 4 | 3 | 2 | 1 | 0 |

call Hypercall `call` (5.2.1) permitted if set.

4.1.4.5 Semaphore Capability

A [Semaphore Capability](#) ($CAP_{OBJ_{SM}}$) references a semaphore (2.5). The capability permissions are defined as follows:

| | | | | |
|---|---|---|----|----|
| ~ | ~ | ~ | dn | up |
| 4 | 3 | 2 | 1 | 0 |

up Hypercall `sm_ctrl[up]` (5.4.3) permitted if set.

dn Hypercall `sm_ctrl[down]` (5.4.3) permitted if set.

4.1.5 Reply Capability

A [Reply Capability](#) (CAP_{RP}) references a caller execution context. It is stored in the reply register of an execution context during communication and automatically destroyed when invoked.

4.2 Capability Selector

A [Capability Selector](#) (SEL) is a user-visible abstract key for accessing a capability. The capability selector serves as integer index for the memory space, port I/O space, or object space of a protection domain. All capability selectors that do not refer to capabilities of another type refer to a null capability. For example, in Figure 4.1 capability selector 2 refers to a capability for an execution context.

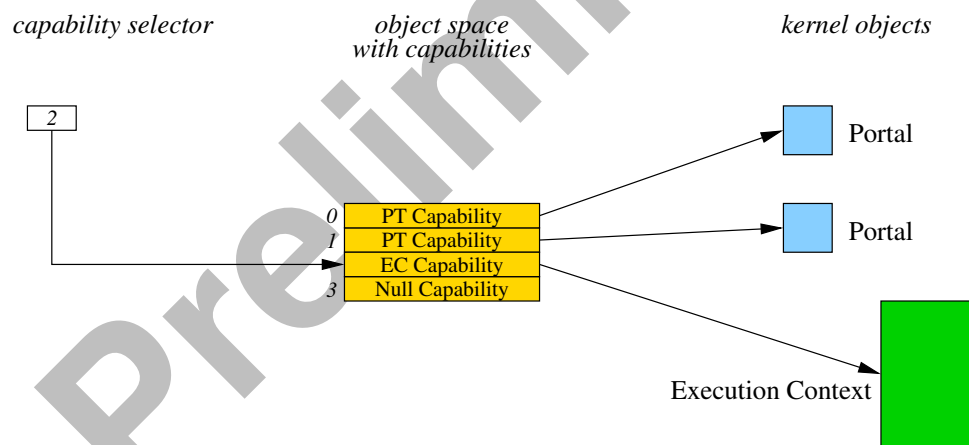


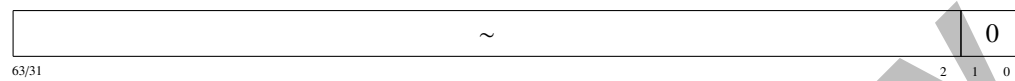
Figure 4.1: Capability Selector

4.3 Capability Range Descriptor

A [Capability Range Descriptor \(CRD\)](#) refers to all capabilities of a particular type in the selector range $SEL \dots SEL + 2^{\text{Order}} - 1$. It must be naturally aligned such that $SEL \equiv 0 \pmod{2^{\text{Order}}}$. During capability delegation, the permissions of the destination capability are computed as the logical AND of the permissions of the source capability, the permission mask from the send capability range descriptor, and the permission mask from the receive capability range descriptor.

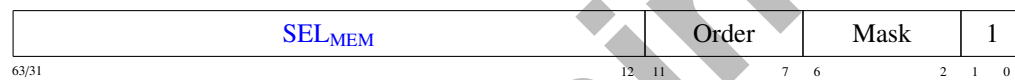
4.3.1 Null Capability Range Descriptor

A [Null Capability Range Descriptor \(CRD₀\)](#) does not refer to any capabilities.



4.3.2 Memory Capability Range Descriptor

A [Memory Capability Range Descriptor \(CRD_{MEM}\)](#) refers to the memory capabilities located within the specified selector range of the memory space. Each memory capability covers 2^{12} bytes of memory.



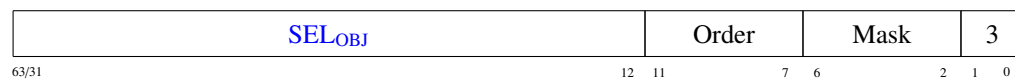
4.3.3 Port I/O Capability Range Descriptor

An [Port I/O Capability Range Descriptor \(CRD_{PIO}\)](#) refers to the port I/O capabilities located within the specified selector range of the port I/O space.



4.3.4 Object Capability Range Descriptor

An [Object Capability Range Descriptor \(CRD_{OBJ}\)](#) refers to the object capabilities located within the specified selector range of the object space.



4.4 Message Transfer Descriptor

The [Message Transfer Descriptor \(MTD\)](#) is an architecture-specific bitfield that controls the contents of an exception or intercept message. The [MTD](#) is provided by the portal associated with the event and conveyed to the receiver as part of the exception or intercept message.

For each bit set to 1, the microhypervisor transfers the processor state associated with that bit to/from the respective fields of the [UTCB](#) data area. The layout of the [MTD](#) and the fields in the [UTCB](#) data area are described in the processor-specific [ABI \(IV\)](#).

4.5 Quantum Priority Descriptor

The [Quantum Priority Descriptor \(QPD\)](#) specifies the priority of a scheduling context and its time quantum in microseconds. It has the following format:

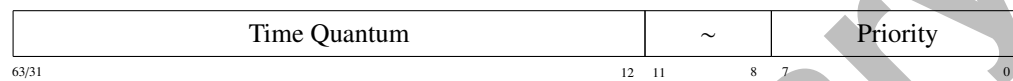


Figure 4.2: Quantum Priority Descriptor

4.6 User Thread Control Block

All execution contexts, except virtual CPUs, have their own private **User Thread Control Block (UTCB)**, which consists of a header area and a data area as illustrated in Figure 4.3.

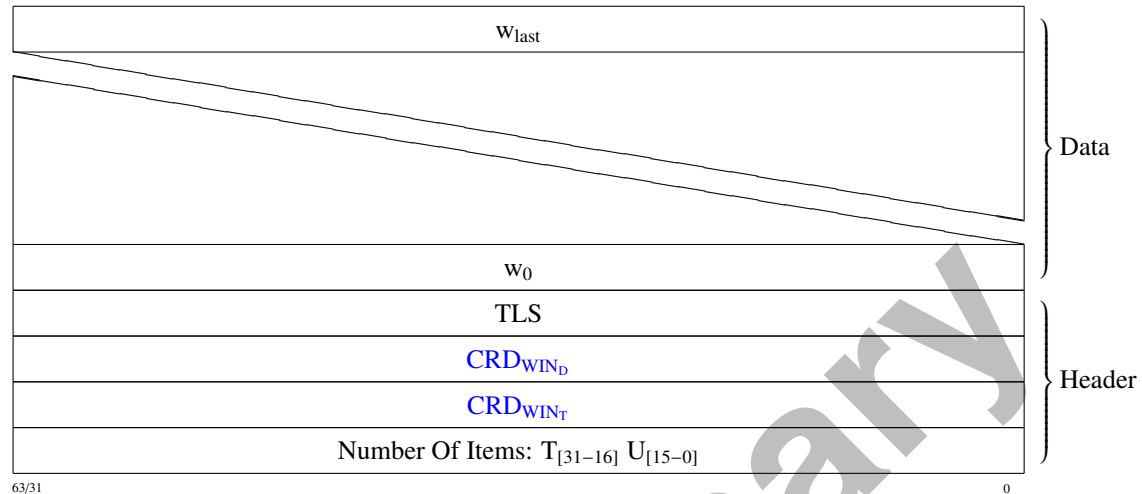


Figure 4.3: User Thread Control Block: General Layout

4.6.1 Header Area

The **UTCB** header fields are defined as follows:

U

Number of untyped items.

T

Number of typed items.

CRD_{WIN_T}

This capability range descriptor (4.3) specifies a receive window in the memory, port I/O, or object space, in which the microhypervisor is allowed to perform capability translations. A null capability range descriptor disables capability translations.

CRD_{WIN_D}

This capability range descriptor (4.3) specifies a receive window in the memory, port I/O, or object space, in which the execution context is willing to accept capability delegations. A null capability range descriptor disables capability delegations.

TLS

This field is never written by the microhypervisor and can be used to store thread-local data.

4.6.2 Data Area

The size of the data area is defined by the size of the **UTCB** minus the size of the header area. An execution context uses its **UTCB** to send or receive messages, and to transfer typed items during capability delegation. The U and T fields in the **UTCB** header area define the number of untyped and typed items.

4.6.2.1 Untyped Items

The microhypervisor transfers untyped items from the beginning of the **UTCB** data area upwards. Each untyped item occupies one word as illustrated in Figure 4.4. For example, during a transfer of u untyped items, the microhypervisor copies words $w_0 \dots w_{u-1}$ from the **UTCB** data area of the sender to words $w_0 \dots w_{u-1}$ in the **UTCB** data area of the receiver, without interpreting the contents of these words.

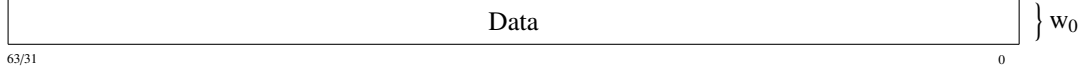


Figure 4.4: User Thread Control Block: Untyped Item

4.6.2.2 Typed Items

The microhypervisor transfers typed items from the end of the **UTCB** data area downwards. Each typed item occupies two words. For example, during a transfer of t typed items, the microhypervisor interprets words $w_{\text{last}} \dots w_{\text{last}-2t+1}$ of the sender's **UTCB** data area. For each typed item in the sender **UTCB**, the microhypervisor creates a corresponding typed item in the receiver **UTCB**. The following typed items are currently defined:

Translate:



Figure 4.5: User Thread Control Block: Translate Item

If the type of the sender's **CRD** does not match the type of the receive window $\text{CRD}_{\text{WIN}_T}$ in the receiver's **UTCB** header, the receiver obtains a typed item with a null capability range descriptor.

Otherwise, the microhypervisor attempts to translate the capability range specified by the base address and order in the sender protection domain to the corresponding capability range in the receiver protection domain from which it had been originally delegated. If the translation fails, e.g., because the sender range is not derived from the receiver range, the receiver obtains a typed item with a null capability range descriptor. Otherwise the capability range descriptor describes the corresponding range in the receiver and the sender permissions for that range.

Delegate:

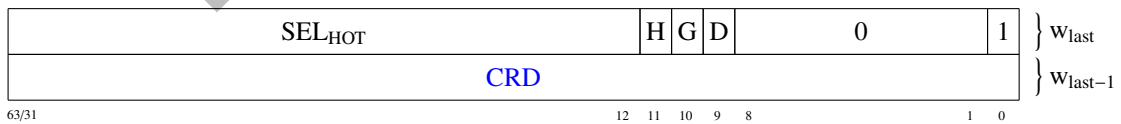


Figure 4.6: User Thread Control Block: Delegate Item

If the type of the sender's **CRD** does not match the type of the receive window $\text{CRD}_{\text{WIN}_D}$ in the receiver's **UTCB** header, the receiver obtains a typed item with a null capability range descriptor.

Otherwise, the microhypervisor computes the range of capabilities to delegate from the sender to the receiver, using the hotspot SEL_{HOT} for range disambiguation, as described in Section 3.5. The capability range descriptor in the receiver's typed item describes the contents of the receive window.

The following bits provide additional control over the capability delegation:

H

If the bit is clear, the source of the capability delegation is the protection domain itself. If the bit is set, the source is set to the microhypervisor. Only the root protection domain can use this bit to request capabilities from the microhypervisor. For other protection domains this bit is treated as clear.

G

If the bit is clear, the resources referenced by the [CRD](#) will not be accessible in the guest VM. If the bit is set, the resources will be accessible in the guest VM. This bit is only applicable to [CRD_{MEM}](#) and [CRD_{PIO}](#).

D

If the bit is clear, the resources referenced by the [CRD](#) will not be accessible by DMA. If the bit is set, the resources will be accessible by DMA. This bit is only applicable to [CRD_{MEM}](#).

Preliminary

5 Hypercalls

5.1 Definitions

Hypercall Numbers

Each hypercall is identified by a unique number. Figure 5.1 lists the currently defined hypercalls.

| Number | Hypercall | Section |
|--------|------------|---------|
| 0x0 | CALL | 5.2.1 |
| 0x1 | REPLY | 5.2.2 |
| 0x2 | CREATE_PD | 5.3.1 |
| 0x3 | CREATE_EC | 5.3.2 |
| 0x4 | CREATE_SC | 5.3.3 |
| 0x5 | CREATE_PT | 5.3.4 |
| 0x6 | CREATE_SM | 5.3.5 |
| 0x7 | REVOKE | 5.3.6 |
| 0x8 | LOOKUP | 5.3.7 |
| 0x9 | EC_CTRL | 5.4.1 |
| 0xa | SC_CTRL | 5.4.2 |
| 0xb | SM_CTRL | 5.4.3 |
| 0xc | ASSIGN_PCI | 5.5.1 |
| 0xd | ASSIGN_GSI | 5.5.2 |

Figure 5.1: Hypercall Numbers

Status Codes

Figure 5.2 shows the status codes returned to indicate success or failure of a hypercall.

| Number | Status Code | Description |
|--------|-------------|-----------------------|
| 0x0 | SUCCESS | Successful Operation |
| 0x1 | COM_TIM | Communication Timeout |
| 0x2 | COM_ABT | Communication Abort |
| 0x3 | BAD_HYP | Invalid Hypercall |
| 0x4 | BAD_CAP | Invalid Capability |
| 0x5 | BAD_PAR | Invalid Parameter |
| 0x6 | BAD_FTR | Invalid Feature |
| 0x7 | BAD_CPU | Invalid CPU Number |
| 0x8 | BAD_DEV | Invalid Device ID |

Figure 5.2: Status Codes

5.2 Communication

5.2.1 Call

Synopsis:

```
status = call (SEL_OBJ_PT);
```

Parameters:

SEL_OBJ_PT: Target Portal

Flags:

| | | |
|---|----|-----|
| 0 | DD | DB |
| 3 | 2 | 1 0 |

DB Disable Blocking (0=blocking, 1=nonblocking)

DD Disable Donation (0=dcall, 1=ncall)

Description:

1. If the execution context (2.2), to which the target portal referenced by **SEL_OBJ_PT** is bound, is busy, the microhypervisor considers the 'disable blocking' flag. If the flag is set, the hypercall returns with a timeout. Otherwise the caller blocks until the callee execution context becomes available.
2. The microhypervisor transfers a message, whose contents is determined by the **UTCB**, from the caller to the callee.
3. The microhypervisor establishes a reply capability (4.1) in the reply register of the callee. The caller blocks until the callee invokes the reply capability. If the 'disable donation' flag is clear, the current scheduling context, previously bound to the caller, is donated and thereby bound to the callee.

Status:

SUCCESS

Hypercall completed successfully.

COM_TIM

Rendezvous with the callee execution context timed out.

COM_ABT

Operation aborted during execution of the callee execution context.

BAD_CAP

SEL_OBJ_PT did not refer to a **Portal Capability (CAP_OBJ_PT)**.

BAD_CPU

Caller execution context and callee execution context are on different CPUs.

5.2.2 Reply

Synopsis:

```
SELOBJPT = reply();
```

Description:

1. If the reply register contains a reply capability, the microhypervisor transfers a message, whose contents is determined by the **UTCB**, to the caller execution context referenced by the reply capability.
2. If the caller had donated its scheduling context to the callee, the microhypervisor binds that scheduling context back to the caller, thereby terminating the donation.
3. The microhypervisor destroys the reply capability by replacing it with a null capability CAP_0 .
4. The callee blocks until a subsequent request arrives.

Status:

This hypercall does not return. Instead, when one of the portals bound to the execution context is called, the microhypervisor passes the capability selector of the called portal to the execution context, and execution continues at the instruction pointer specified in that portal.

5.3 Capability Management

5.3.1 Create Protection Domain

Synopsis:

```
status = create_pd (SELOBJ0, SELOBJPD, CRDOBJ);
```

Parameters:

SEL_{OBJ₀}: Created PD

SEL_{OBJ_{PD}}: Owner PD

CRD_{OBJ}: Initial Portals

Description:

Creates a new protection domain, accounted to the PD specified by SEL_{OBJ_{PD}}. Prior to the hypercall, SEL_{OBJ₀} must refer to a null capability, and SEL_{OBJ_{PD}} must refer to a protection domain capability with permission bit CAP_{PD} set. The caller PD obtains in place of SEL_{OBJ₀} a protection domain capability that refers to the created PD. The microhypervisor delegates the capability range, specified by CRD_{OBJ}, from the caller PD to the created PD.

Status:

SUCCESS

Hypercall completed successfully.

BAD_CAP

SEL_{OBJ₀} did not refer to a Null Capability (CAP₀).

SEL_{OBJ_{PD}} did not refer to a Protection Domain Capability (CAP_{OBJ_{PD}}).

CAP_{OBJ_{PD}} has insufficient permissions.

5.3.2 Create Execution Context

Synopsis:

```
status = create_ec (SEL_OBJ0, SEL_OBJPD, CPU, UTCB, SP, SEL_EVT);
```

Parameters:

SEL_OBJ₀: Created EC

SEL_OBJ_{PD}: Owner PD

CPU: CPU Number

UTCB: Virtual Address: UTCB Pointer

SP: Virtual Address: Initial Stack Pointer

SEL_EVT: Event Selector Base

Flags:

| | |
|---|-----|
| 0 | G |
| 3 | 1 0 |

G Global Thread (0=local, 1=global)

Description:

Creates a new execution context, accounted to the PD specified by **SEL_OBJ_{PD}**, and sets the processor affinity according to CPU. Prior to the hypercall, **SEL_OBJ₀** must refer to a null capability, and **SEL_OBJ_{PD}** must refer to a protection domain capability with permission bit CAP_{EC} set. The caller PD obtains in place of **SEL_OBJ₀** an execution context capability that refers to the created EC. The microhypervisor binds the execution context to the protection domain referred to by **SEL_OBJ_{PD}** in the caller PD.

If the UTCB address is zero, the microhypervisor creates a virtual CPU, otherwise it creates a thread according to the G flag. Local threads cannot have a scheduling context bound to them. Their initial state is as if they had just done a **reply** hypercall, so they start running when they receive a request on a portal bound to them. Global threads and virtual CPUs generate a startup exception the first time a scheduling context is bound to them.

The hypervisor sets the initial stack pointer only once during creation of the execution context. Subsequently the execution context is responsible for maintaining its stack pointer across hypercalls. Applications can also use the initial stack pointer value as a means to identify execution contexts during their startup exception.

Status:

SUCCESS

Hypercall completed successfully.

BAD_CAP

SEL_OBJ₀ did not refer to a **Null Capability** (CAP₀).

SEL_OBJ_{PD} did not refer to a **Protection Domain Capability** (CAP_{OBJ_{PD}}).

CAP_{OBJ_{PD}} has insufficient permissions.

BAD_CPU

Invalid CPU number.

BAD_FTR

Virtual CPUs not supported.

BAD_PAR

Invalid UTCB address.

5.3.3 Create Scheduling Context

Synopsis:

```
status = create_sc (SEL_OBJ0, SEL_OBJPD, SEL_OBJEC, QPD);
```

Parameters:

SEL_OBJ₀: Created SC

SEL_OBJ_{PD}: Owner PD

SEL_OBJ_{EC}: Bound EC

QPD: Quantum Priority Descriptor (4.5)

Description:

Creates a new scheduling context, accounted to the PD specified by SEL_OBJ_{PD}, and sets the scheduling parameters according to QPD. Prior to the hypercall, SEL_OBJ₀ must refer to a null capability, SEL_OBJ_{PD} must refer to a protection domain capability with permission bit CAP_{SC} set, and SEL_OBJ_{EC} must refer to an execution context capability with permission bit CAP_{SC} set. The caller PD obtains in place of SEL_OBJ₀ a scheduling context capability that refers to the created SC. The microhypervisor binds the scheduling context to the execution context referred to by SEL_OBJ_{EC} in the caller PD.

Status:

SUCCESS

Hypercall completed successfully.

BAD_CAP

SEL_OBJ₀ did not refer to a Null Capability (CAP₀).

SEL_OBJ_{PD} did not refer to a Protection Domain Capability (CAP_{OBJ_{PD}}).

SEL_OBJ_{EC} did not refer to an Execution Context Capability (CAP_{OBJ_{EC}}).

CAP_{OBJ_{PD}} or CAP_{OBJ_{EC}} has insufficient permissions.

Binding the scheduling context to the execution context failed.

BAD_PAR

QPD time quantum or priority is zero.

5.3.4 Create Portal

Synopsis:

```
status = create_pt (SEL_OBJ0, SEL_OBJPD, SEL_OBJEC, MTD, IP);
```

Parameters:

SEL_{OBJ₀}: Created PT

SEL_{OBJ_{PD}}: Owner PD

SEL_{OBJ_{EC}}: Bound EC

MTD: Message Transfer Descriptor (4.4)

IP: Virtual Address: Instruction Pointer

Description:

Creates a new portal, accounted to the PD specified by SEL_{OBJ_{PD}}. Prior to the hypercall, SEL_{OBJ₀} must refer to a null capability, SEL_{OBJ_{PD}} must refer to a protection domain capability with permission bit CAP_{PT} set, and SEL_{OBJ_{EC}} must refer to an execution context capability with permission bit CAP_{PT} set. The caller PD obtains in place of SEL_{OBJ₀} a portal capability that refers to the created portal. The microhypervisor binds the portal to the execution context referred to by SEL_{OBJ_{EC}} in the caller PD.

Status:

SUCCESS

Hypercall completed successfully.

BAD_CAP

SEL_{OBJ₀} did not refer to a **Null Capability** (CAP₀).

SEL_{OBJ_{PD}} did not refer to a **Protection Domain Capability** (CAP_{OBJ_{PD}}).

SEL_{OBJ_{EC}} did not refer to an **Execution Context Capability** (CAP_{OBJ_{EC}}).

CAP_{OBJ_{PD}} or CAP_{OBJ_{EC}} has insufficient permissions.

Binding the portal to the execution context failed.

5.3.5 Create Semaphore

Synopsis:

```
status = create_sm (SEL_OBJ0, SEL_OBJPD, CNT);
```

Parameters:

SEL_OBJ₀: Created SM

SEL_OBJ_{PD}: Owner PD

CNT: Unsigned: Initial Counter Value

Description:

Creates a new semaphore, accounted to the PD specified by SEL_OBJ_{PD}. Prior to the hypercall, SEL_OBJ₀ must refer to a null capability, and SEL_OBJ_{PD} must refer to a protection domain capability with permission bit CAP_{SM} set. The caller PD obtains in place of SEL_OBJ₀ a semaphore capability that refers to the created semaphore. The microhypervisor initializes the semaphore counter with the value of CNT.

Status:

SUCCESS

Hypercall completed successfully.

BAD_CAP

SEL_OBJ₀ did not refer to a Null Capability (CAP₀).

SEL_OBJ_{PD} did not refer to a Protection Domain Capability (CAP_{OBJ_{PD}}).

CAP_{OBJ_{PD}} has insufficient permissions.

5.3.6 Revoke Capability Range

Synopsis:

```
status = revoke (CRD);
```

Parameters:

CRD: Capability Range Descriptor (4.3)

Flags:

| | |
|---|-----|
| 0 | SR |
| 3 | 1 0 |

SR Self Revoke (0=only children, 1=including self)

Description:

Revokes permissions from all inherited capabilities in the range specified by the [Capability Range Descriptor \(CRD\)](#). If the self revoke bit is set, the permissions will also be revoked from the range specified by the [Capability Range Descriptor \(CRD\)](#). See Section 3.6 for more details.

This operation never fails but can take a long time to complete if there are many capabilities to revoke.

Status:

SUCCESS

Hypercall completed successfully.

5.3.7 Lookup Capability Range

Synopsis:

```
status = lookup (CRD);
```

Parameters:

CRD: Capability Range Descriptor (4.3)

Description:

Looks up a range of capabilities in the caller's protection domain. The caller must specify a base address and type in the CRD prior to the hypercall. If a capability exists at the specified address, the microhypervisor returns a completely filled CRD describing the capability range. Otherwise a null capability range descriptor is returned.

Status:

SUCCESS

Hypercall completed successfully.

5.4 Execution Control

5.4.1 Execution Context Control

Synopsis:

```
status = ec_ctrl (SELOBJEC);
```

Parameters:

SEL_{OBJEC}: Execution Context

Description:

Pends an event for the specified execution context, which causes it to generate a recall exception before its next return from the microhypervisor.

Status:

SUCCESS

Hypercall completed successfully.

BAD_CAP

SEL_{OBJEC} did not refer to an Execution Context Capability (CAP_{OBJEC}).
CAP_{OBJEC} has insufficient permissions.

5.4.2 Scheduling Context Control

Synopsis:

```
status = sc_ctrl (SEL_OBJsc, &Time);
```

Parameters:

SEL_OBJ_{sc}: Scheduling Context

Return Values:

Time: Aggregate consumed execution time in microseconds.

Description:

Returns runtime statistics for the specified scheduling context.

Status:

SUCCESS

Hypercall completed successfully.

BAD_CAP

SEL_OBJ_{sc} did not refer to a Scheduling Context Capability (CAP_OBJ_{sc}).
CAP_OBJ_{sc} has insufficient permissions.

5.4.3 Semaphore Control

Synopsis:

```
status = sm_ctrl (SEL_OBJSM);
```

Parameters:

SEL_OBJ_{SM}: Semaphore

Flags:

| | | |
|---|----|-----|
| 0 | ZC | OP |
| 3 | 2 | 1 0 |

OP Operation (0=up, 1=down)

ZC Zero Counter (0=decrement, 1=set to zero)

Description:

The *down* operation blocks the calling execution context if the semaphore counter is zero, otherwise the counter is decremented or set to zero, depending on the setting of the ZC bit.

The *up* operation releases an execution context blocked on the semaphore if one exists, otherwise it increments the counter.

Status:

SUCCESS

Hypercall completed successfully.

BAD_CAP

SEL_OBJ_{SM} did not refer to a Semaphore Capability (CAP_OBJ_{SM}).

CAP_OBJ_{SM} has insufficient permissions.

5.5 Device Control

5.5.1 Assign PCI Device

Synopsis:

```
status = assign_pci (SEL_OBJPD, SEL_MEMDEV, RID);
```

Parameters:

SEL_{OBJ_{PD}}: Target PD

SEL_{MEM_{DEV}}: PCI Device

RID: Routing Hint

Description:

Assigns the PCI device, named by **SEL_{MEM_{DEV}}**, to the protection domain, named by **SEL_{OBJ_{PD}}**. **SEL_{MEM_{DEV}}** must refer to a memory capability for the memory-mapped PCI configuration space of the device.

Status:

SUCCESS

Hypercall completed successfully.

BAD_CAP

SEL_{OBJ_{PD}} did not refer to a **Protection Domain Capability** (**CAP_{OBJ_{PD}}**).

BAD_DEV

SEL_{MEM_{DEV}} did not refer to a valid PCI device.

5.5.2 Assign Global System Interrupt

Synopsis:

```
status = assign_gsi (SEL_OBJSM, SEL_MEMDEV, CPU, &MSI);
```

Parameters:

SEL_OBJ_{SM}: Interrupt Semaphore

SEL_MEM_{DEV}: PCI or HPET Device

CPU: CPU Number

Return Values:

MSI: Values to program into the MSI registers of the PCI or HPET device to ensure proper operation.

Description:

Routes the Global System Interrupt (GSI), named by **SEL_OBJ_{SM}**, to the specified **CPU**, where it will be signaled on the corresponding interrupt semaphore. For Message Signaled Interrupts, **SEL_MEM_{DEV}** must refer to a memory capability that names the device, according to the following table:

| Interrupt Type | Device Type | SEL_MEM_{DEV} | MSI Hint |
|-----------------------------|-------------|---|----------|
| Message Signaled Interrupt | PCI HPET | PCI Configuration Space Device I/O Space | Valid |
| IOAPIC Interrupt Pin | Any | Ignored | N/A |

Status:

SUCCESS

Hypercall completed successfully.

BAD_CAP

SEL_OBJ_{SM} did not refer to an Interrupt Semaphore Capability (**CAP_OBJ_{SM}**).

BAD_DEV

SEL_MEM_{DEV} did not refer to a valid PCI or HPET device.

BAD_CPU

Invalid CPU number.

6 Booting

6.1 Root Protection Domain

When the microhypervisor has initialized the system, it creates the root protection domain with a root execution context and a root scheduling context.

6.1.1 Resource Access

Execution contexts in the root protection domain have the special ability to request resources from the microhypervisor during communication, by setting the H-bit in a typed item (4.6.2.2). In addition to memory and I/O ports, the following capabilities can be requested:

Idle Scheduling Contexts

Capability selectors 0 ... $n - 1$ in the microhypervisor refer to $CAP_{OBJ_{sc}}$ for the idle thread of the respective CPU, where n is the maximum number of supported CPUs, as indicated by the HIP. These capabilities can be used with the `sc_ctrl` hypercall.

Interrupt Semaphores

Capability selectors $n \dots n + GSI - 1$ in the microhypervisor refer to $CAP_{OBJ_{sm}}$ for global system interrupts, where GSI is the maximum number of supported GSIs, as indicated by the HIP. These capabilities can be used with the `sm_ctrl` and `assign_gsi` hypercalls.

6.1.2 Initial Configuration

At bootup the root protection domain is configured as follows:

6.1.2.1 Memory Space

Program Segments

The microhypervisor loads the program segments of the roottask into the memory space as specified by the ELF program headers of the roottask image.

Hypervisor Information Page

The HIP is mapped into the memory space at a specific virtual address that is passed to the root execution context during startup.

UTCB

The UTCB of the root execution context is mapped into the memory space just below the HIP.

All other regions of the memory space are initially empty.

6.1.2.2 Port I/O Space

The port I/O space is initially empty.

6.1.2.3 Object Space

The object space contains the following capabilities:

- Capability selector EXC + 0 refers to the root PD capability.
- Capability selector EXC + 1 refers to the root EC capability.
- Capability selector EXC + 2 refers to the root SC capability.

All other capability selectors refer to null capabilities.

6.2 Hypervisor Information Page

The [Hypervisor Information Page \(HIP\)](#) conveys information about the platform and configuration to the root protection domain. The processor register that contains the virtual address of the [HIP](#) during booting is ABI-specific ([IV](#)). Figure 6.1 shows the layout of the [Hypervisor Information Page](#). All fields are unsigned values unless stated otherwise.

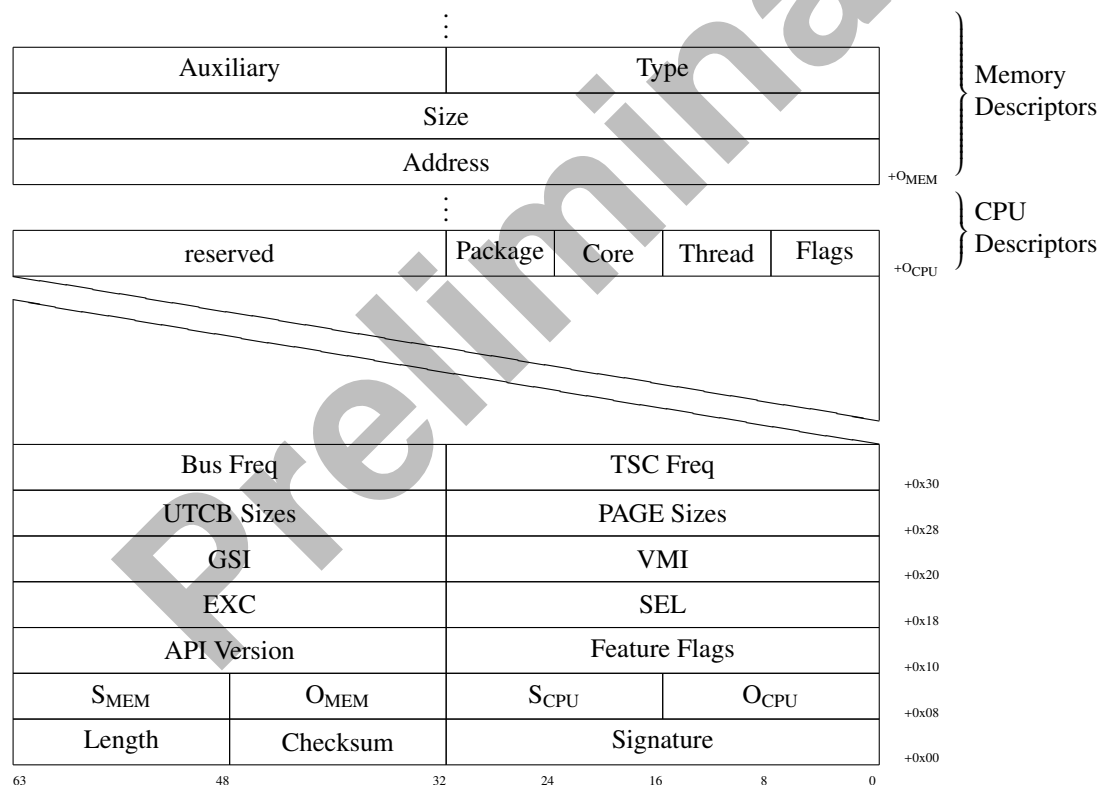


Figure 6.1: Hypervisor Information Page

Signature:

A value of 0x41564f4e identifies the NOVA microhypervisor.

Checksum:

The checksum is valid if 16bit-wise addition the [HIP](#) contents produces a value of 0.

Length:

Length of the [HIP](#) in bytes. This includes all CPU and memory descriptors.

O_{CPU}:

Offset of the first CPU descriptor in bytes, relative to the [HIP](#) base.

S_{CPU}:

Size of one CPU descriptor in bytes.

O_{MEM}:

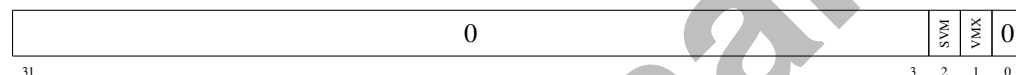
Offset of the first memory descriptor in bytes, relative to the [HIP](#) base.

S_{MEM}:

Size of one memory descriptor in bytes.

Feature Flags:

The microhypervisor supports a particular feature if and only if the corresponding bit in the feature flags is set to 1. The following features are currently defined:



VMX: Intel Virtual Machine Extensions

SVM: AMD Secure Virtual Machine

API Version:

API version number.

SEL:

Number of available capability selectors in each object space. Specifying a capability selector beyond the maximum number supported wraps around to the beginning of the object space.

EXC:

Number of capability selectors used for exception handling ([3.3](#)).

VMI:

Number of capability selectors used for virtual-machine intercept handling ([3.3](#)).

GSI:

Number of global system interrupts ([3.4](#)).

PAGE Sizes:

If bit *n* is set, the implementation supports memory pages of size 2^n bytes.

UTCB Sizes:

If bit *n* is set, the implementation supports user thread control blocks of size 2^n bytes.

TSC Freq:

Time Stamp Counter Frequency in kHz.

BUS Freq:

Interconnect Frequency in kHz.

CPU Descriptor

The array of CPU descriptors contains n_{cpu} entries, where

$$n_{cpu} = \frac{O_{MEM} - O_{CPU}}{S_{CPU}}. \quad (6.1)$$

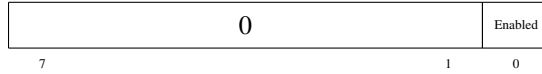
The value of n_{cpu} reflects the maximum number of CPUs supported by the microhypervisor. The array index of a CPU descriptor corresponds to the CPU number that must be specified for certain hypercalls to target that CPU. A CPU can only be used if its descriptor is marked as enabled.

Package, Core, Thread:

CPU multiprocessor topology information.

Flags:

CPU status flags.



MEM Descriptor

The array of MEM descriptors contains n_{mem} entries, where

$$n_{mem} = \frac{Length - O_{MEM}}{S_{MEM}}. \quad (6.2)$$

Memory descriptors with a positive type field provide information about the memory layout of the platform, which corresponds to the memory map provided by firmware. User applications should ignore or truncate memory descriptors for regions outside the addressable range, e.g., physical memory beyond 4GB for 32bit APIs.

Memory descriptors with a negative type field provide information about preallocated regions of physical memory. User applications should assume that the preallocated regions overlap the physical memory regions of the platform.

Address:

Physical base address of memory region.

Size:

Size of memory region in bytes.

Type:

The following types of memory region are currently defined:

| Type | Description | |
|-------|--------------------------|---------------------------|
| 1 | Available Memory | Platform Physical Memory |
| 2 | Reserved Memory | |
| 3 | ACPI Reclaim Memory | |
| 4 | ACPI NVS Memory | |
| Other | Treat as Reserved Memory | |
| -1 | Microhypervisor | Allocated Physical Memory |
| -2 | Multiboot Module | |

Auxiliary:

Physical address of command line if type is 'Multiboot Module', reserved otherwise.

Part IV

Application Binary Interface

7 ABI x86-32

7.1 Initial State

Figure 7.1 details the state of the CPU registers when the microhypervisor has finished booting and transfers control to the root protection domain.

| Register | Description |
|-------------------------|--|
| CS | Selector=~, Base=0, Limit=0xFFFFFFFF, Code Segment, ro |
| SS,DS,ES,FS,GS | Selector=~, Base=0, Limit=0xFFFFFFFF, Data Segment, rw |
| EIP | Address of entry point from ELF header |
| ESP | Address of hypervisor information page |
| EAX | Bootstrap CPU number |
| ECX,EDX,EBX,EBP,ESI,EDI | ~ |
| EFLAGS | 0x202 |

Figure 7.1: Initial State

7.2 Event-Specific Capability Selectors

For the delivery of exception and intercept messages, the microhypervisor performs an implicit portal traversal. The selector for the destination portal (SEL_{OBJPT}) is determined by adding the exception number or VM exit reason to SEL_{EVT} of the affected execution context.

Exceptions

| SEL_{OBJPT} | Exception | SEL_{OBJPT} | Exception |
|-------------------|------------------|--------------------|------------------|
| $SEL_{EVT} + 0x0$ | #DE | $SEL_{EVT} + 0x10$ | #MF |
| $SEL_{EVT} + 0x1$ | #DB | $SEL_{EVT} + 0x11$ | #AC |
| $SEL_{EVT} + 0x2$ | reserved | $SEL_{EVT} + 0x12$ | #MC ¹ |
| $SEL_{EVT} + 0x3$ | #BP | $SEL_{EVT} + 0x13$ | #XM |
| $SEL_{EVT} + 0x4$ | #OF | $SEL_{EVT} + 0x14$ | reserved |
| $SEL_{EVT} + 0x5$ | #BR | $SEL_{EVT} + 0x15$ | reserved |
| $SEL_{EVT} + 0x6$ | #UD | $SEL_{EVT} + 0x16$ | reserved |
| $SEL_{EVT} + 0x7$ | #NM ¹ | $SEL_{EVT} + 0x17$ | reserved |
| $SEL_{EVT} + 0x8$ | #DF ¹ | $SEL_{EVT} + 0x18$ | reserved |
| $SEL_{EVT} + 0x9$ | reserved | $SEL_{EVT} + 0x19$ | reserved |
| $SEL_{EVT} + 0xa$ | #TS ¹ | $SEL_{EVT} + 0x1a$ | reserved |
| $SEL_{EVT} + 0xb$ | #NP | $SEL_{EVT} + 0x1b$ | reserved |
| $SEL_{EVT} + 0xc$ | #SS | $SEL_{EVT} + 0x1c$ | reserved |
| $SEL_{EVT} + 0xd$ | #GP | $SEL_{EVT} + 0x1d$ | reserved |
| $SEL_{EVT} + 0xe$ | #PF | $SEL_{EVT} + 0x1e$ | STARTUP |
| $SEL_{EVT} + 0xf$ | reserved | $SEL_{EVT} + 0x1f$ | RECALL |

VMX Intercepts

| Number | Intercept | Number | Intercept | Number | Intercept |
|--------|-------------------------------|--------|----------------------------------|--------|----------------------------|
| 0x0 | Exception or NMI ¹ | 0x15 | VMPTRLD | 0x2a | reserved |
| 0x1 | INTR ¹ | 0x16 | VMPTRST | 0x2b | TPR Below Threshold |
| 0x2 | Triple Fault ² | 0x17 | VMREAD | 0x2c | APIC Access |
| 0x3 | INIT ² | 0x18 | VMRESUME | 0x2d | reserved |
| 0x4 | SIPI ² | 0x19 | VMWRITE | 0x2e | GDTR/IDTR Access |
| 0x5 | I/O SMI | 0x1a | VMXOFF | 0x2f | LDTR/TR Access |
| 0x6 | Other SMI | 0x1b | VMXON | 0x30 | EPT Violation ² |
| 0x7 | Interrupt Window | 0x1c | CR Access ¹ | 0x31 | EPT Misconfiguration |
| 0x8 | NMI Window | 0x1d | DR Access | 0x32 | INVEPT |
| 0x9 | Task Switch ² | 0x1e | I/O Access ² | 0x33 | RDTSCP |
| 0xa | CPUID ² | 0x1f | RDMSR ² | 0x34 | VMX Preemption Timer |
| 0xb | GETSEC ² | 0x20 | WRMSR ² | 0x35 | INVVPID |
| 0xc | HLT ² | 0x21 | Invalid Guest State ² | 0x36 | WBINVD |
| 0xd | INVD ² | 0x22 | MSR Load Failure | 0x37 | XSETBV |
| 0xe | INVLPG ¹ | 0x23 | reserved | 0x38 | reserved |
| 0xf | RDPMSR | 0x24 | MWAIT | 0x39 | RDRAND |
| 0x10 | RDTSC | 0x25 | MTF | 0x3a | INVPCID |
| 0x11 | RSM | 0x26 | reserved | 0x3b | VMFUNC |
| 0x12 | VMCALL | 0x27 | MONITOR | 0x3c | reserved |
| 0x13 | VMCLEAR | 0x28 | PAUSE | 0xfe | STARTUP |
| 0x14 | VMLAUNCH | 0x29 | Machine Check | 0xff | RECALL |

Please refer to [3] for more details on each of these events.

SVM Intercepts

| Number | Intercept | Number | Intercept | Number | Intercept |
|-----------|------------------------|--------|---------------------------|--------|----------------------------------|
| 0x0–0xf | CR Read ¹ | 0x6e | RDTSC | 0x81 | VMMCALL |
| 0x10–0x1f | CR Write ¹ | 0x6f | RDPMSR | 0x82 | VMLOAD ² |
| 0x20–0x2f | DR Read | 0x70 | PUSHF | 0x83 | VMSAVE ² |
| 0x30–0x3f | DR Write | 0x71 | POPF | 0x84 | STGI |
| 0x40–0x5f | Exception ¹ | 0x72 | CPUID | 0x85 | CLGI ² |
| 0x60 | INTR ¹ | 0x73 | RSM | 0x86 | SKINIT ² |
| 0x61 | NMI ¹ | 0x74 | IRET | 0x87 | RDTSCP |
| 0x62 | SMI | 0x75 | INT | 0x88 | ICEBP |
| 0x63 | INIT ² | 0x76 | INVD ² | 0x89 | WBINVD |
| 0x64 | Interrupt Window | 0x77 | PAUSE | 0x8a | MONITOR |
| 0x65 | CR0 Selective Write | 0x78 | HLT ² | 0x8b | MWAIT |
| 0x66 | IDTR Read | 0x79 | INVLPG ¹ | 0x8c | MWAIT (cond.) |
| 0x67 | GDTR Read | 0x7a | INVLPGA | 0x8d | reserved |
| 0x68 | LDTR Read | 0x7b | I/O Access ² | 0x8e | reserved |
| 0x69 | TR Read | 0x7c | MSR Access ² | 0x8f | reserved |
| 0x6a | IDTR Write | 0x7d | Task Switch | 0xfc | NPT Fault ² |
| 0x6b | GDTR Write | 0x7e | FERR Freeze | 0xfd | Invalid Guest State ² |
| 0x6c | LDTR Write | 0x7f | Triple Fault ² | 0xfe | STARTUP |
| 0x6d | TR Write | 0x80 | VMRUN | 0xff | RECALL |

Please refer to [1] for more details on each of these events.

Preliminary

¹These events may be handled by the microhypervisor, in which case they will not cause portal traversals.

²These events may be force-enabled by the microhypervisor, in which case they will cause portal traversals.

7.3 UTCB Data Area Layout

| TSC Offset | | TSC Value | | |
|---------------------|-----------------------|---------------------|------------------------|----------|
| reserved | IDTR Base | IDTR Limit | reserved | |
| reserved | GDTR Base | GDTR Limit | reserved | |
| reserved | TR Base | TR Limit | TR AR | TR Sel |
| reserved | LDTR Base | LDTR Limit | LDTR AR | LDTR Sel |
| reserved | GS Base | GS Limit | GS AR | GS Sel |
| reserved | FS Base | FS Limit | FS AR | FS Sel |
| reserved | DS Base | DS Limit | DS AR | DS Sel |
| reserved | SS Base | SS Limit | SS AR | SS Sel |
| reserved | CS Base | CS Limit | CS AR | CS Sel |
| reserved | ES Base | ES Limit | ES AR | ES Sel |
| SYSENTER EIP | SYSENTER ESP | SYSENTER CS | DR7 | |
| CR4 | CR3 | CR2 | CR0 | |
| Preemption Timer | | Secondary Exit Ctrl | Primary Exit Ctrl | |
| Secondary Exit Qual | | Primary Exit Qual | | |
| EDI | ESI | EBP | ESP | |
| EBX | EDX | ECX | EAX | |
| Injection Error | Injection Information | Activity State | Interruptibility State | |
| EFLAGS | EIP | Instruction Length | MTD | |

Format of Injection Information

| V | ~ | | | | | | | | | | N | I | E | Type | Vector | | | |
|----|----|--|--|--|--|--|--|--|--|--|----|----|----|------|--------|---|---|---|
| 31 | 30 | | | | | | | | | | 14 | 13 | 12 | 11 | 10 | 8 | 7 | 0 |

Vector

IDT Vector of Interrupt or Exception

Type

- 0 = External Interrupt
- 2 = Non-Maskable Interrupt
- 3 = Hardware Exception
- 4 = Software Interrupt
- 5 = Privileged Software Exception
- 6 = Software Exception

E

- 0 = Do not deliver the error code from the *Injection Error* field of the UTCB
- 1 = Deliver the error code from the *Injection Error* field of the UTCB

I

- 0 = Do not request an interrupt window
- 1 = Request an interrupt window

N

- 0 = Do not request an NMI window
- 1 = Request an NMI window

V

- 0 = Injection Information fields *Vector*, *Type*, *E* are invalid
- 1 = Injection Information fields *Vector*, *Type*, *E* are valid

Format of Segment Access Rights

| | | | | | | | | | | | | |
|----|----|----|-----|----|-----|---|-----|---|------|---|---|---|
| ~ | U | G | D/B | L | AVL | P | DPL | S | Type | | | |
| 15 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 0 |

Type

Segment Type

S

Descriptor Type:
0 = System
1 = Code or Data

DPL

Descriptor Privilege Level

P

Segment Present

AVL

Available for use by system software

L

64-bit mode active (CS only)

D/B

Default Operation Size:
0 = 16-bit segment
1 = 32-bit segment

G

Granularity

U

Segment Unusable:
0 = Segment Usable
1 = Segment Unusable

7.4 Message Transfer Descriptor

The [MTD](#), which controls the state transfer for exceptions and intercepts, as described in [Section 4.4](#), has the following format:

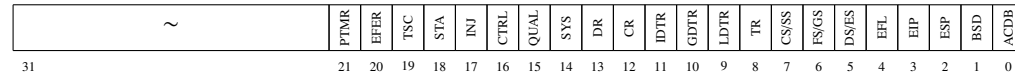


Figure 7.2: Message Transfer Descriptor

| Bit | Type | Exceptions | Intercepts |
|-------|------|----------------------------------|---|
| ACDB | rw | EAX, ECX, EDX, EBX | EAX, ECX, EDX, EBX |
| BSD | rw | EBP, ESI, EDI | EBP, ESI, EDI |
| ESP | rw | ESP | ESP |
| EIP | rw | EIP | EIP, Instruction Length |
| EFL | rw | EFLAGS ¹ | EFLAGS |
| DS/ES | rw | ≡ | DS, ES (Selector, Base, Limit, Access Rights) |
| FS/GS | rw | ≡ | FS, GS (Selector, Base, Limit, Access Rights) |
| CS/SS | rw | ≡ | CS, SS (Selector, Base, Limit, Access Rights) |
| TR | rw | ≡ | TR (Selector, Base, Limit, Access Rights) |
| LDTR | rw | ≡ | LDTR (Selector, Base, Limit, Access Rights) |
| GDTR | rw | ≡ | GDTR (Base, Limit) |
| IDTR | rw | ≡ | IDTR (Base, Limit) |
| CR | rw | ≡ | CR0, CR2, CR3, CR4 |
| DR | rw | ≡ | DR7 |
| SYS | rw | ≡ | SYSENTER MSRs (CS, ESP, EIP) |
| QUAL | r | Exit Qualifications ² | Exit Qualifications |
| CTRL | w | ≡ | Execution Controls |
| INJ | rw | ≡ | Injection Info, Injection Error Code |
| STA | rw | ≡ | Interruptibility State, Activity State |
| TSC | rw | ≡ | TSC Value, TSC Offset ³ |
| EFER | rw | ≡ | EFER MSR |
| PTMR | rw | ≡ | Preemption Timer |

Each bit controls the transfer of a subset of the [CPU](#) state to/from the respective [UTCB](#) fields ([7.3](#)). State with access type *r* can be read from [CPU](#) into the [UTCB](#). State with access type *w* can be written from the [UTCB](#) into the [CPU](#).

¹Only the arithmetic flags are writable.

²The primary exit qualification contains the exception error code. The secondary exit qualification contains the fault address.

³Reads load the absolute value of the TSC offset into the UTCB. Writes add the value from UTCB to the TSC offset.

7.5 Calling Convention

The following pages describes the calling convention for each hypercall. An execution context calls into the microhypervisor by loading the hypercall identifier and other parameters into the specified processor registers and then executes the *sysenter* instruction.

The hypercall identifier consists of the hypercall number and hypercall-specific flags, as illustrated in Figure 7.3.

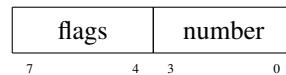


Figure 7.3: Hypercall Identifier

The status code returned from a hypercall has the format shown in Figure 7.4.

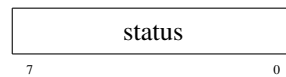
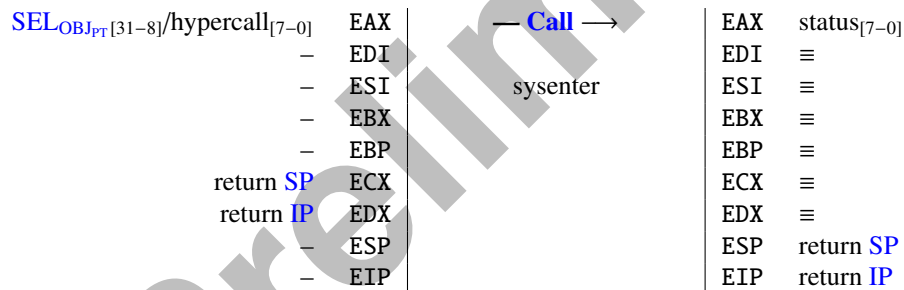


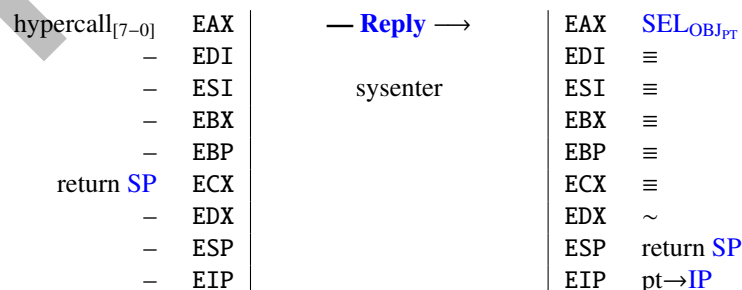
Figure 7.4: Status Code

The assignment of hypercall parameters to general-purpose registers is shown on the left side; the contents of the registers after the hypercall is shown on the right side.

Call



Reply



Create Protection Domain

| | | | | |
|------------------------------------|-----|----------------------|-----|-------------------------|
| $SEL_{OBJ_0}[31-8]/hypercall[7-0]$ | EAX | — Create PD → | EAX | status _[7-0] |
| $SEL_{OBJ_{PD}}$ | EDI | | EDI | ≡ |
| CRD_{OBJ} | ESI | sysenter | ESI | ≡ |
| — | EBX | | EBX | ≡ |
| — | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

Create Execution Context

| | | | | |
|------------------------------------|-----|----------------------|-----|-------------------------|
| $SEL_{OBJ_0}[31-8]/hypercall[7-0]$ | EAX | — Create EC → | EAX | status _[7-0] |
| $SEL_{OBJ_{PD}}$ | EDI | | EDI | ≡ |
| $UTCB[31-12]/CPU[11-0]$ | ESI | sysenter | ESI | ≡ |
| SP | EBX | | EBX | ≡ |
| SEL_{EVT} | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

Create Scheduling Context

| | | | | |
|------------------------------------|-----|----------------------|-----|-------------------------|
| $SEL_{OBJ_0}[31-8]/hypercall[7-0]$ | EAX | — Create SC → | EAX | status _[7-0] |
| $SEL_{OBJ_{PD}}$ | EDI | | EDI | ≡ |
| $SEL_{OBJ_{EC}}$ | ESI | sysenter | ESI | ≡ |
| QPD | EBX | | EBX | ≡ |
| — | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

Create Portal

| | | | | |
|------------------------------------|-----|----------------------|-----|-------------------------|
| $SEL_{OBJ_0}[31-8]/hypercall[7-0]$ | EAX | — Create PT → | EAX | status _[7-0] |
| $SEL_{OBJ_{PD}}$ | EDI | | EDI | ≡ |
| $SEL_{OBJ_{EC}}$ | ESI | sysenter | ESI | ≡ |
| MTD | EBX | | EBX | ≡ |
| IP | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

Create Semaphore

| | | | | |
|---|-----|----------------------|-----|-------------------------|
| $SEL_{OBJ_0[31-8]}$ /hypercall _[7-0] | EAX | — Create SM → | EAX | status _[7-0] |
| $SEL_{OBJ_{PD}}$ | EDI | | EDI | ≡ |
| CNT | ESI | sysenter | ESI | ≡ |
| — | EBX | | EBX | ≡ |
| — | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

Revoke Capability Range

| | | | | |
|----------------------------|-----|-------------------|-----|-------------------------|
| hypercall _[7-0] | EAX | — Revoke → | EAX | status _[7-0] |
| CRD | EDI | | EDI | ≡ |
| — | ESI | sysenter | ESI | ≡ |
| — | EBX | | EBX | ≡ |
| — | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

Lookup Capability Range

| | | | | |
|----------------------------|-----|-------------------|-----|-------------------------|
| hypercall _[7-0] | EAX | — Lookup → | EAX | status _[7-0] |
| CRD | EDI | | EDI | CRD |
| — | ESI | sysenter | ESI | ≡ |
| — | EBX | | EBX | ≡ |
| — | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

Execution Context Control

| | | | | |
|--|-----|-----------------------|-----|-------------------------|
| $SEL_{OBJ_{EC}[31-8]}$ /hypercall _[7-0] | EAX | — EC Control → | EAX | status _[7-0] |
| — | EDI | | EDI | ≡ |
| — | ESI | sysenter | ESI | ≡ |
| — | EBX | | EBX | ≡ |
| — | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

Scheduling Context Control

| | | | | |
|--|-----|-----------------------|-----|-------------------------|
| SEL _{OBJ_{SC}} _[31-8] /hypercall _[7-0] | EAX | — SC Control → | EAX | status _[7-0] |
| — | EDI | | EDI | Time _[63-32] |
| — | ESI | sysenter | ESI | Time _[31-0] |
| — | EBX | | EBX | ≡ |
| — | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

Semaphore Control

| | | | | |
|--|-----|-----------------------|-----|-------------------------|
| SEL _{OBJ_{SM}} _[31-8] /hypercall _[7-0] | EAX | — SM Control → | EAX | status _[7-0] |
| — | EDI | | EDI | ≡ |
| — | ESI | sysenter | ESI | ≡ |
| — | EBX | | EBX | ≡ |
| — | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

Assign PCI Device

| | | | | |
|--|-----|-----------------------|-----|-------------------------|
| SEL _{OBJ_{PD}} _[31-8] /hypercall _[7-0] | EAX | — Assign PCI → | EAX | status _[7-0] |
| SEL _{MEM_{DEV}} | EDI | | EDI | ≡ |
| RID | ESI | sysenter | ESI | ≡ |
| — | EBX | | EBX | ≡ |
| — | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

Assign Global System Interrupt

| | | | | |
|--|-----|-----------------------|-----|-------------------------|
| SEL _{OBJ_{SM}} _[31-8] /hypercall _[7-0] | EAX | — Assign GSI → | EAX | status _[7-0] |
| SEL _{MEM_{DEV}} | EDI | | EDI | MSI Addr |
| CPU | ESI | sysenter | ESI | MSI Data |
| — | EBX | | EBX | ≡ |
| — | EBP | | EBP | ≡ |
| return SP | ECX | | ECX | ≡ |
| return IP | EDX | | EDX | ≡ |
| — | ESP | | ESP | return SP |
| — | EIP | | EIP | return IP |

8 ABI x86-64

8.1 Initial State

Figure 8.1 details the state of the **CPU** registers when the microhypervisor has finished booting and transfers control to the root protection domain.

| Register | Description |
|-------------------------------|--|
| CS | Selector=~, Base=0, Limit=0xFFFFFFFF, Code Segment, ro |
| SS,DS,ES,FS,GS | Selector=~, Base=0, Limit=0xFFFFFFFF, Data Segment, rw |
| RIP | Address of entry point from ELF header |
| RSP | Address of hypervisor information page |
| RDI | Bootstrap CPU number |
| RAX,RCX,RDX,RBX,RBP,RSI | ~ |
| R8,R9,R10,R11,R12,R13,R14,R15 | ~ |
| RFLAGS | 0x202 |

Figure 8.1: Initial State

8.2 Event-Specific Capability Selectors

For the delivery of exception and intercept messages, the microhypervisor performs an implicit portal traversal. The selector for the destination portal (SEL_{OBJPT}) is determined by adding the exception number or VM exit reason to SEL_{EVT} of the affected execution context.

Exceptions

| SEL_{OBJPT} | Exception | SEL_{OBJPT} | Exception |
|-------------------|------------------|--------------------|------------------|
| $SEL_{EVT} + 0x0$ | #DE | $SEL_{EVT} + 0x10$ | #MF |
| $SEL_{EVT} + 0x1$ | #DB | $SEL_{EVT} + 0x11$ | #AC |
| $SEL_{EVT} + 0x2$ | reserved | $SEL_{EVT} + 0x12$ | #MC ¹ |
| $SEL_{EVT} + 0x3$ | #BP | $SEL_{EVT} + 0x13$ | #XM |
| $SEL_{EVT} + 0x4$ | #OF | $SEL_{EVT} + 0x14$ | reserved |
| $SEL_{EVT} + 0x5$ | #BR | $SEL_{EVT} + 0x15$ | reserved |
| $SEL_{EVT} + 0x6$ | #UD | $SEL_{EVT} + 0x16$ | reserved |
| $SEL_{EVT} + 0x7$ | #NM ¹ | $SEL_{EVT} + 0x17$ | reserved |
| $SEL_{EVT} + 0x8$ | #DF ¹ | $SEL_{EVT} + 0x18$ | reserved |
| $SEL_{EVT} + 0x9$ | reserved | $SEL_{EVT} + 0x19$ | reserved |
| $SEL_{EVT} + 0xa$ | #TS ¹ | $SEL_{EVT} + 0x1a$ | reserved |
| $SEL_{EVT} + 0xb$ | #NP | $SEL_{EVT} + 0x1b$ | reserved |
| $SEL_{EVT} + 0xc$ | #SS | $SEL_{EVT} + 0x1c$ | reserved |
| $SEL_{EVT} + 0xd$ | #GP | $SEL_{EVT} + 0x1d$ | reserved |
| $SEL_{EVT} + 0xe$ | #PF | $SEL_{EVT} + 0x1e$ | STARTUP |
| $SEL_{EVT} + 0xf$ | reserved | $SEL_{EVT} + 0x1f$ | RECALL |

VMX Intercepts

| Number | Intercept | Number | Intercept | Number | Intercept |
|--------|-------------------------------|--------|----------------------------------|--------|----------------------------|
| 0x0 | Exception or NMI ¹ | 0x15 | VMPTRLD | 0x2a | reserved |
| 0x1 | INTR ¹ | 0x16 | VMPTRST | 0x2b | TPR Below Threshold |
| 0x2 | Triple Fault ² | 0x17 | VMREAD | 0x2c | APIC Access |
| 0x3 | INIT ² | 0x18 | VMRESUME | 0x2d | reserved |
| 0x4 | SIPI ² | 0x19 | VMWRITE | 0x2e | GDTR/IDTR Access |
| 0x5 | I/O SMI | 0x1a | VMXOFF | 0x2f | LDTR/TR Access |
| 0x6 | Other SMI | 0x1b | VMXON | 0x30 | EPT Violation ² |
| 0x7 | Interrupt Window | 0x1c | CR Access ¹ | 0x31 | EPT Misconfiguration |
| 0x8 | NMI Window | 0x1d | DR Access | 0x32 | INVEPT |
| 0x9 | Task Switch ² | 0x1e | I/O Access ² | 0x33 | RDTSCP |
| 0xa | CPUID ² | 0x1f | RDMSR ² | 0x34 | VMX Preemption Timer |
| 0xb | GETSEC ² | 0x20 | WRMSR ² | 0x35 | INVVPID |
| 0xc | HLT ² | 0x21 | Invalid Guest State ² | 0x36 | WBINVD |
| 0xd | INVD ² | 0x22 | MSR Load Failure | 0x37 | XSETBV |
| 0xe | INVLPG ¹ | 0x23 | reserved | 0x38 | reserved |
| 0xf | RDPMSR | 0x24 | MWAIT | 0x39 | RDRAND |
| 0x10 | RDTSC | 0x25 | MTF | 0x3a | INVPCID |
| 0x11 | RSM | 0x26 | reserved | 0x3b | VMFUNC |
| 0x12 | VMCALL | 0x27 | MONITOR | 0x3c | reserved |
| 0x13 | VMCLEAR | 0x28 | PAUSE | 0xfe | STARTUP |
| 0x14 | VMLAUNCH | 0x29 | Machine Check | 0xff | RECALL |

Please refer to [3] for more details on each of these events.

SVM Intercepts

| Number | Intercept | Number | Intercept | Number | Intercept |
|-----------|------------------------|--------|---------------------------|--------|----------------------------------|
| 0x0–0xf | CR Read ¹ | 0x6e | RDTSC | 0x81 | VMMCALL |
| 0x10–0x1f | CR Write ¹ | 0x6f | RDPMSR | 0x82 | VMLOAD ² |
| 0x20–0x2f | DR Read | 0x70 | PUSHF | 0x83 | VMSAVE ² |
| 0x30–0x3f | DR Write | 0x71 | POPF | 0x84 | STGI |
| 0x40–0x5f | Exception ¹ | 0x72 | CPUID | 0x85 | CLGI ² |
| 0x60 | INTR ¹ | 0x73 | RSM | 0x86 | SKINIT ² |
| 0x61 | NMI ¹ | 0x74 | IRET | 0x87 | RDTSCP |
| 0x62 | SMI | 0x75 | INT | 0x88 | ICEBP |
| 0x63 | INIT ² | 0x76 | INVD ² | 0x89 | WBINVD |
| 0x64 | Interrupt Window | 0x77 | PAUSE | 0x8a | MONITOR |
| 0x65 | CR0 Selective Write | 0x78 | HLT ² | 0x8b | MWAIT |
| 0x66 | IDTR Read | 0x79 | INVLPG ¹ | 0x8c | MWAIT (cond.) |
| 0x67 | GDTR Read | 0x7a | INVLPGA | 0x8d | reserved |
| 0x68 | LDTR Read | 0x7b | I/O Access ² | 0x8e | reserved |
| 0x69 | TR Read | 0x7c | MSR Access ² | 0x8f | reserved |
| 0x6a | IDTR Write | 0x7d | Task Switch | 0xfc | NPT Fault ² |
| 0x6b | GDTR Write | 0x7e | FERR Freeze | 0xfd | Invalid Guest State ² |
| 0x6c | LDTR Write | 0x7f | Triple Fault ² | 0xfe | STARTUP |
| 0x6d | TR Write | 0x80 | VMRUN | 0xff | RECALL |

Please refer to [1] for more details on each of these events.

Preliminary

¹These events may be handled by the microhypervisor, in which case they will not cause portal traversals.

²These events may be force-enabled by the microhypervisor, in which case they will cause portal traversals.

8.3 UTCB Data Area Layout

| | | | | | |
|---------------------|-----------------------|---------------------|------------------------|----------|--------|
| TSC Offset | | TSC Value | | +0x1c0 | |
| IDTR Base | | IDTR Limit | reserved | | +0x1b0 |
| GDTR Base | | GDTR Limit | reserved | | +0x1a0 |
| TR Base | | TR Limit | TR AR | TR Sel | +0x190 |
| LDTR Base | | LDTR Limit | LDTR AR | LDTR Sel | +0x180 |
| GS Base | | GS Limit | GS AR | GS Sel | +0x170 |
| FS Base | | FS Limit | FS AR | FS Sel | +0x160 |
| DS Base | | DS Limit | DS AR | DS Sel | +0x150 |
| SS Base | | SS Limit | SS AR | SS Sel | +0x140 |
| CS Base | | CS Limit | CS AR | CS Sel | +0x130 |
| ES Base | | ES Limit | ES AR | ES Sel | +0x120 |
| SYSENTER RIP | | SYSENTER RSP | | | +0x110 |
| SYSENTER CS | | DR7 | | | +0x100 |
| EFER | | CR8 | | | +0xf0 |
| CR4 | | CR3 | | | +0xe0 |
| CR2 | | CR0 | | | +0xd0 |
| Preemption Timer | | Secondary Exit Ctrl | Primary Exit Ctrl | | +0xc0 |
| Secondary Exit Qual | | Primary Exit Qual | | | +0xb0 |
| R15 | | R14 | | | +0xa0 |
| R13 | | R12 | | | +0x90 |
| R11 | | R10 | | | +0x80 |
| R9 | | R8 | | | +0x70 |
| RDI | | RSI | | | +0x60 |
| RBP | | RSP | | | +0x50 |
| RBX | | RDX | | | +0x40 |
| RCX | | RAX | | | +0x30 |
| Injection Error | Injection Information | Activity State | Interruptibility State | | +0x20 |
| RFLAGS | | RIP | | | +0x10 |
| Instruction Length | | MTD | | | +0x00 |

Format of Injection Information

| | | | | | | | | | | | | | | | | | | | | | | |
|----|----|--|--|--|--|--|--|--|--|--|--|--|--|----|----|----|----|------|--------|---|---|--|
| V | ~ | | | | | | | | | | | | | | N | I | E | Type | Vector | | | |
| 31 | 30 | | | | | | | | | | | | | 14 | 13 | 12 | 11 | 10 | 8 | 7 | 0 | |

Vector

IDT Vector of Interrupt or Exception

Type

- 0 = External Interrupt
- 2 = Non-Maskable Interrupt
- 3 = Hardware Exception
- 4 = Software Interrupt
- 5 = Privileged Software Exception
- 6 = Software Exception

E

- 0 = Do not deliver the error code from the *Injection Error* field of the UTCB
- 1 = Deliver the error code from the *Injection Error* field of the UTCB

I

- 0 = Do not request an interrupt window
- 1 = Request an interrupt window

N

- 0 = Do not request an NMI window
- 1 = Request an NMI window

V

- 0 = Injection Information fields *Vector*, *Type*, *E* are invalid
- 1 = Injection Information fields *Vector*, *Type*, *E* are valid

Format of Segment Access Rights

| | | | | | | | | | | | | |
|----|----|----|-----|----|-----|---|-----|---|------|---|---|---|
| ~ | U | G | D/B | L | AVL | P | DPL | S | Type | | | |
| 15 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 0 |

Type

Segment Type

S

Descriptor Type:
 0 = System
 1 = Code or Data

DPL

Descriptor Privilege Level

P

Segment Present

AVL

Available for use by system software

L

64-bit mode active (CS only)

D/B

Default Operation Size:

0 = 16-bit segment

1 = 32-bit segment

G

Granularity

U

Segment Unusable:

0 = Segment Usable

1 = Segment Unusable

Preliminary

8.4 Message Transfer Descriptor

The [MTD](#), which controls the state transfer for exceptions and intercepts, as described in [Section 4.4](#), has the following format:

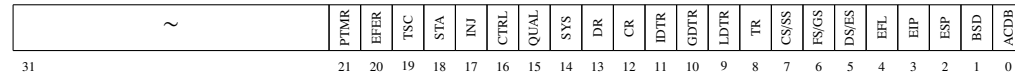


Figure 8.2: Message Transfer Descriptor

| Bit | Type | Exceptions | Intercepts |
|-------|------|----------------------------------|---|
| ACDB | rw | EAX, ECX, EDX, EBX | EAX, ECX, EDX, EBX |
| BSD | rw | EBP, ESI, EDI | EBP, ESI, EDI |
| ESP | rw | ESP | ESP |
| EIP | rw | EIP | EIP, Instruction Length |
| EFL | rw | EFLAGS ¹ | EFLAGS |
| DS/ES | rw | ≡ | DS, ES (Selector, Base, Limit, Access Rights) |
| FS/GS | rw | ≡ | FS, GS (Selector, Base, Limit, Access Rights) |
| CS/SS | rw | ≡ | CS, SS (Selector, Base, Limit, Access Rights) |
| TR | rw | ≡ | TR (Selector, Base, Limit, Access Rights) |
| LDTR | rw | ≡ | LDTR (Selector, Base, Limit, Access Rights) |
| GDTR | rw | ≡ | GDTR (Base, Limit) |
| IDTR | rw | ≡ | IDTR (Base, Limit) |
| CR | rw | ≡ | CR0, CR2, CR3, CR4 |
| DR | rw | ≡ | DR7 |
| SYS | rw | ≡ | SYSENTER MSRs (CS, ESP, EIP) |
| QUAL | r | Exit Qualifications ² | Exit Qualifications |
| CTRL | w | ≡ | Execution Controls |
| INJ | rw | ≡ | Injection Info, Injection Error Code |
| STA | rw | ≡ | Interruptibility State, Activity State |
| TSC | rw | ≡ | TSC Value, TSC Offset ³ |
| EFER | rw | ≡ | EFER MSR |
| PTMR | rw | ≡ | Preemption Timer |

Each bit controls the transfer of a subset of the [CPU](#) state to/from the respective [UTCB](#) fields ([7.3](#)). State with access type *r* can be read from [CPU](#) into the [UTCB](#). State with access type *w* can be written from the [UTCB](#) into the [CPU](#).

¹Only the arithmetic flags are writable.

²The primary exit qualification contains the exception error code. The secondary exit qualification contains the fault address.

³Reads load the absolute value of the TSC offset into the UTCB. Writes add the value from UTCB to the TSC offset.

8.5 Calling Convention

The following pages describes the calling convention for each hypercall. An execution context calls into the microhypervisor by loading the hypercall identifier and other parameters into the specified processor registers and then executes the *syscall* instruction.

The hypercall identifier consists of the hypercall number and hypercall-specific flags, as illustrated in Figure 8.3.



Figure 8.3: Hypercall Identifier

The status code returned from a hypercall has the format shown in Figure 8.4.

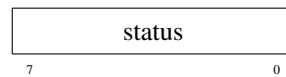
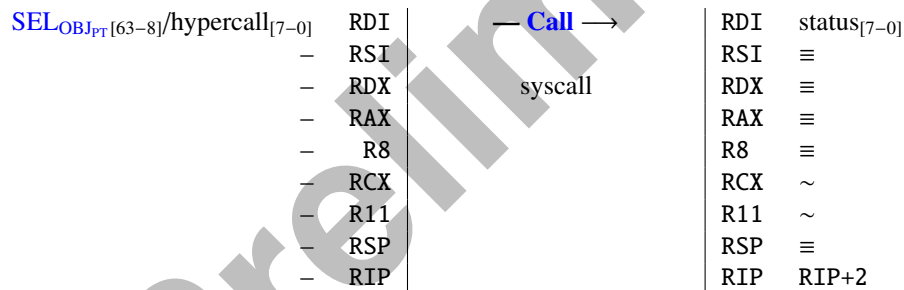


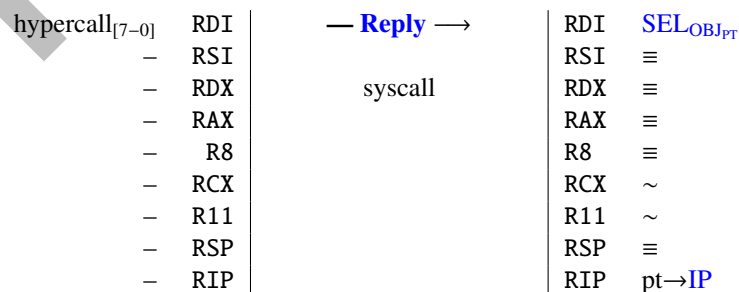
Figure 8.4: Status Code

The assignment of hypercall parameters to general-purpose registers is shown on the left side; the contents of the registers after the hypercall is shown on the right side.

Call



Reply



Create Protection Domain

| | | | | |
|------------------------------------|-----|----------------------|-----|-------------------------|
| $SEL_{OBJ_0}[63-8]/hypercall[7-0]$ | RDI | — Create PD → | RDI | status _[7-0] |
| $SEL_{OBJ_{PD}}$ | RSI | | RSI | ≡ |
| CRD_{OBJ} | RDX | syscall | RDX | ≡ |
| — | RAX | | RAX | ≡ |
| — | R8 | | R8 | ≡ |
| — | RCX | | RCX | ~ |
| — | R11 | | R11 | ~ |
| — | RSP | | RSP | ≡ |
| — | RIP | | RIP | RIP+2 |

Create Execution Context

| | | | | |
|------------------------------------|-----|----------------------|-----|-------------------------|
| $SEL_{OBJ_0}[63-8]/hypercall[7-0]$ | RDI | — Create EC → | RDI | status _[7-0] |
| $SEL_{OBJ_{PD}}$ | RSI | | RSI | ≡ |
| $UTCB[63-12]/CPU[11-0]$ | RDX | syscall | RDX | ≡ |
| SP | RAX | | RAX | ≡ |
| SEL_{EVT} | R8 | | R8 | ≡ |
| — | RCX | | RCX | ~ |
| — | R11 | | R11 | ~ |
| — | RSP | | RSP | ≡ |
| — | RIP | | RIP | RIP+2 |

Create Scheduling Context

| | | | | |
|------------------------------------|-----|----------------------|-----|-------------------------|
| $SEL_{OBJ_0}[63-8]/hypercall[7-0]$ | RDI | — Create SC → | RDI | status _[7-0] |
| $SEL_{OBJ_{PD}}$ | RSI | | RSI | ≡ |
| $SEL_{OBJ_{EC}}$ | RDX | syscall | RDX | ≡ |
| QPD | RAX | | RAX | ≡ |
| — | R8 | | R8 | ≡ |
| — | RCX | | RCX | ~ |
| — | R11 | | R11 | ~ |
| — | RSP | | RSP | ≡ |
| — | RIP | | RIP | RIP+2 |

Create Portal

| | | | | |
|------------------------------------|-----|----------------------|-----|-------------------------|
| $SEL_{OBJ_0}[63-8]/hypercall[7-0]$ | RDI | — Create PT → | RDI | status _[7-0] |
| $SEL_{OBJ_{PD}}$ | RSI | | RSI | ≡ |
| $SEL_{OBJ_{EC}}$ | RDX | syscall | RDX | ≡ |
| MTD | RAX | | RAX | ≡ |
| IP | R8 | | R8 | ≡ |
| — | RCX | | RCX | ~ |
| — | R11 | | R11 | ~ |
| — | RSP | | RSP | ≡ |
| — | RIP | | RIP | RIP+2 |

Create Semaphore

| | | | | |
|---------------------------------------|-----|----------------------|-----|-------------------------|
| $SEL_{OBJ_0}[63-8]/hypercall_{[7-0]}$ | RDI | — Create SM → | RDI | status _[7-0] |
| $SEL_{OBJ_{PD}}$ | RSI | | RSI | ≡ |
| CNT | RDX | syscall | RDX | ≡ |
| — | RAX | | RAX | ≡ |
| — | R8 | | R8 | ≡ |
| — | RCX | | RCX | ~ |
| — | R11 | | R11 | ~ |
| — | RSP | | RSP | ≡ |
| — | RIP | | RIP | RIP+2 |

Revoke Capability Range

| | | | | |
|----------------------------|-----|-------------------|-----|-------------------------|
| hypercall _[7-0] | RDI | — Revoke → | RDI | status _[7-0] |
| CRD | RSI | | RSI | ≡ |
| — | RDX | syscall | RDX | ≡ |
| — | RAX | | RAX | ≡ |
| — | R8 | | R8 | ≡ |
| — | RCX | | RCX | ~ |
| — | R11 | | R11 | ~ |
| — | RSP | | RSP | ≡ |
| — | RIP | | RIP | RIP+2 |

Lookup Capability Range

| | | | | |
|----------------------------|-----|-------------------|-----|-------------------------|
| hypercall _[7-0] | RDI | — Lookup → | RDI | status _[7-0] |
| CRD | RSI | | RSI | CRD |
| — | RDX | syscall | RDX | ≡ |
| — | RAX | | RAX | ≡ |
| — | R8 | | R8 | ≡ |
| — | RCX | | RCX | ~ |
| — | R11 | | R11 | ~ |
| — | RSP | | RSP | ≡ |
| — | RIP | | RIP | RIP+2 |

Execution Context Control

| | | | | |
|--|-----|-----------------------|-----|-------------------------|
| $SEL_{OBJ_{EC}}[63-8]/hypercall_{[7-0]}$ | RDI | — EC Control → | RDI | status _[7-0] |
| — | RSI | | RSI | ≡ |
| — | RDX | syscall | RDX | ≡ |
| — | RAX | | RAX | ≡ |
| — | R8 | | R8 | ≡ |
| — | RCX | | RCX | ~ |
| — | R11 | | R11 | ~ |
| — | RSP | | RSP | ≡ |
| — | RIP | | RIP | RIP+2 |

Scheduling Context Control

| | | | | |
|--|-----|-----------------------|-----|-------------------------|
| SEL _{OBJ_{SC}} _[63-8] /hypercall _[7-0] | RDI | — SC Control → | RDI | status _[7-0] |
| — RSI | RSI | | RSI | Time _[63-32] |
| — RDX | RDX | syscall | RDX | Time _[31-0] |
| — RAX | RAX | | RAX | ≡ |
| — R8 | R8 | | R8 | ≡ |
| — RCX | RCX | | RCX | ~ |
| — R11 | R11 | | R11 | ~ |
| — RSP | RSP | | RSP | ≡ |
| — RIP | RIP | | RIP | RIP+2 |

Semaphore Control

| | | | | |
|--|-----|-----------------------|-----|-------------------------|
| SEL _{OBJ_{SM}} _[63-8] /hypercall _[7-0] | RDI | — SM Control → | RDI | status _[7-0] |
| — RSI | RSI | | RSI | ≡ |
| — RDX | RDX | syscall | RDX | ≡ |
| — RAX | RAX | | RAX | ≡ |
| — R8 | R8 | | R8 | ≡ |
| — RCX | RCX | | RCX | ~ |
| — R11 | R11 | | R11 | ~ |
| — RSP | RSP | | RSP | ≡ |
| — RIP | RIP | | RIP | RIP+2 |

Assign PCI Device

| | | | | |
|--|-----|-----------------------|-----|-------------------------|
| SEL _{OBJ_{PD}} _[63-8] /hypercall _[7-0] | RDI | — Assign PCI → | RDI | status _[7-0] |
| SEL _{MEM_{DEV}} | RSI | | RSI | ≡ |
| RID | RDX | syscall | RDX | ≡ |
| — RAX | RAX | | RAX | ≡ |
| — R8 | R8 | | R8 | ≡ |
| — RCX | RCX | | RCX | ~ |
| — R11 | R11 | | R11 | ~ |
| — RSP | RSP | | RSP | ≡ |
| — RIP | RIP | | RIP | RIP+2 |

Assign Global System Interrupt

| | | | | |
|--|-----|-----------------------|-----|-------------------------|
| SEL _{OBJ_{SM}} _[63-8] /hypercall _[7-0] | RDI | — Assign GSI → | RDI | status _[7-0] |
| SEL _{MEM_{DEV}} | RSI | | RSI | MSI Addr |
| CPU | RDX | syscall | RDX | MSI Data |
| — RAX | RAX | | RAX | ≡ |
| — R8 | R8 | | R8 | ≡ |
| — RCX | RCX | | RCX | ~ |
| — R11 | R11 | | R11 | ~ |
| — RSP | RSP | | RSP | ≡ |
| — RIP | RIP | | RIP | RIP+2 |

Part V

Appendix

Preliminary

A Acronyms

| | |
|--------------------------------------|--|
| ABI | Application Binary Interface |
| CAP | Capability |
| CAP₀ | Null Capability |
| CAP_{MEM} | Memory Capability |
| CAP_{OBJ} | Object Capability |
| CAP_{OBJEC} | Execution Context Capability |
| CAP_{OBJPD} | Protection Domain Capability |
| CAP_{OBJPT} | Portal Capability |
| CAP_{OBJSC} | Scheduling Context Capability |
| CAP_{OBJSM} | Semaphore Capability |
| CAP_{PIO} | Port I/O Capability |
| CAP_{RP} | Reply Capability |
| CPU | Central Processing Unit |
| CRD | Capability Range Descriptor |
| CRD₀ | Null Capability Range Descriptor |
| CRD_{MEM} | Memory Capability Range Descriptor |
| CRD_{OBJ} | Object Capability Range Descriptor |
| CRD_{PIO} | Port I/O Capability Range Descriptor |
| CRD_{WIN_D} | Capability Receive Window: Delegation |
| CRD_{WIN_T} | Capability Receive Window: Translation |
| DMA | Direct Memory Access |
| EC | Execution Context |
| ELF | Executable and Linkable Format |
| FPU | Floating Point Unit |
| GSI | Global System Interrupt |
| HIP | Hypervisor Information Page |
| MSI | Message Signaled Interrupt |
| MTD | Message Transfer Descriptor |

| | |
|---------------------------------------|---|
| IOAPIC | I/O Advanced Programmable Interrupt Controller |
| IP | Instruction Pointer |
| PD | Protection Domain |
| PT | Portal |
| QPD | Quantum Priority Descriptor |
| SC | Scheduling Context |
| SEL | Capability Selector |
| SEL_{EVT} | Event Selector Base |
| SEL_{MEM} | Memory Capability Selector |
| SEL_{MEMDEV} | Memory Capability Selector: Device |
| SEL_{OBJ} | Object Capability Selector |
| SEL_{OBJ₀} | Object Capability Selector: Null Capability |
| SEL_{OBJ_{EC}} | Object Capability Selector: Execution Context Capability |
| SEL_{OBJ_{PD}} | Object Capability Selector: Protection Domain Capability |
| SEL_{OBJ_{PT}} | Object Capability Selector: Portal Capability |
| SEL_{OBJ_{SC}} | Object Capability Selector: Scheduling Context Capability |
| SEL_{OBJ_{SM}} | Object Capability Selector: Semaphore Capability |
| SEL_{P_{IO}} | Port I/O Capability Selector |
| SM | Semaphore |
| SP | Stack Pointer |
| UTCB | User Thread Control Block |
| VMM | Virtual-Machine Monitor |
| VM | Virtual Machine |

B Bibliography

- [1] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, 2012. Publication Number: 24593.
- [2] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation. *Advanced Configuration and Power Interface Specification*, 2011. Revision 5.0.
- [3] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B and 3C*, 2012. Order Number: 325462.
- [4] Udo Steinberg and Bernhard Kauer. NOVA: A Microhypervisor-Based Secure Virtualization Architecture. In *Proceedings of the 5th ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 209–222. ACM, 2010.

C Console

The VGA console shows information about the microhypervisor version and architecture, as well as the compiler that was used to build the image. For each physical processor core, the microhypervisor prints information about the topology and the core type. At the bottom of the console, event spinners can optionally be displayed for each core.

```
NOVA Microhypervisor v5-0cb7f70 (x86_32): Aug 3 2012 12:27:17 [gcc 4.8.0]

[ 0] CORE:0:0:0 6:3a:9:1 [12] Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
[ 2] CORE:0:2:0 6:3a:9:1 [12] Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
[ 1] CORE:0:1:0 6:3a:9:1 [12] Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
[ 3] CORE:0:3:0 6:3a:9:1 [12] Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
      ↑↑ Microcode Patch Level
      ↑ ↑↑ ↑ ↑ Family : Model : Stepping : Platform
      ↑ ↑ ↑ Package : Core : Thread
      ↑↑ Core Number

↓ Scheduling Events
↓ Helping Events
  ↓ RCU Grace Periods
    ↓↓ vTLB Fills & vTLB Flushes
      ↓ ... Local APIC Interrupts
        ↓↓ Inter-Processor Interrupts
          ↓↓ ... Global System Interrupts & Message Signaled Interrupts

624 F6 3 01 0123456789ABCDEF0123456789ABCDEF0123456789AB5DEF
214 C2 5 71 0123456789ABCDEF0123456789ABCDEF0123456789ABC5EF
F14 8A 9 13 0123456789ABCDEF0123456789ABCDEF0123456789ABCD3F
AB4 EA C 80 0723456784ABCDEF0123456789ABCDEF0123456789ABCDEA
```

D Download

The source code of the NOVA microhypervisor can be downloaded from GitHub.

<https://github.com/IntelLabs/NOVA>

Preliminary