NOVA Microhypervisor Interface Specification

Udo Steinberg
udo@hypervisor.org

December 8, 2020

Contents

Notation

Throughout this document, the following symbols are used:

∼ Indicates that the value of this parameter or field is **undefined**. Future versions of this specification may define a meaning for the parameter or field.

_ Indicates that the value of this parameter or field is **ignored**. Future versions of this specification may define a meaning for the parameter or field.

≡ Indicates that the value of this parameter or field is **unchanged**. The microhypervisor will preserve the value across hypercalls.

Part I

Introduction

1 System Architecture

The **N**OVA **OS** **V**irtualization **A**rchitecture facilitates the coexistence of multiple legacy guest operating systems and a multi-server user-mode framework on a single platform [8]. The core system leverages virtualization technology provided by modern x86 or ARM platforms and comprises the NOVA microhypervisor and one or more Virtual-Machine Monitors (VMMs).

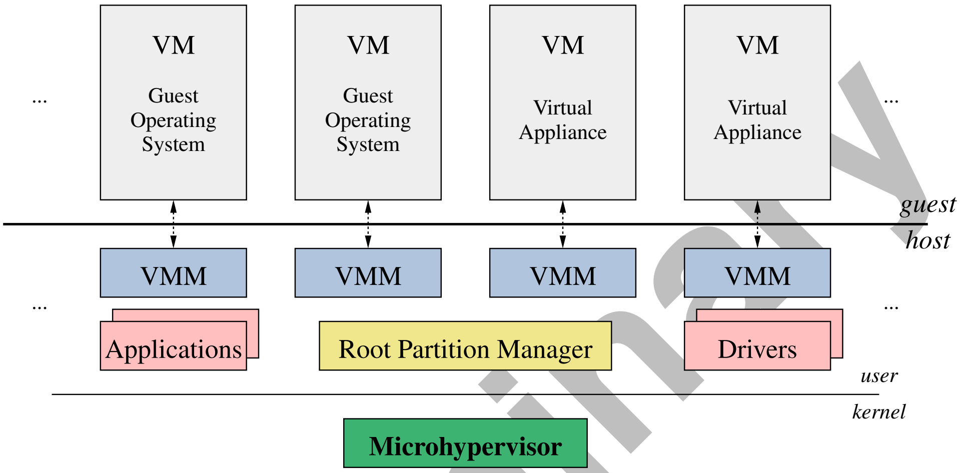


Figure 1.1: System Architecture

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 all applications that do not require virtualization support.

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

Part II

Basic Abstractions

2 Kernel Objects

2.1 Protection Domain

1. The Protection Domain (PD) is a unit of protection and isolation.
2. Access to a Protection Domain (PD) is controlled by a PD Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{PD}}}$).
3. A PD is composed of a set of spaces that store Capabilities (CAP) to kernel objects or platform resources that can be accessed by ECs within that PD. The following subsections detail these spaces.

2.1.1 Object Space

1. Each empty slot of the Object Space ($\mathtt{SPC}_{\mathtt{OBJ}}$) contains a Null Capability ($\mathtt{CAP}_{\emptyset}$).
2. Each non-empty slot of the Object Space ($\mathtt{SPC}_{\mathtt{OBJ}}$) contains an Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}}$) that refers to a kernel object.

2.1.2 Memory Space

1. Each empty slot of the Memory Space ($\mathtt{SPC}_{\mathtt{MEM}}$) contains a Null Capability ($\mathtt{CAP}_{\emptyset}$).
2. Each non-empty slot of the Memory Space ($\mathtt{SPC}_{\mathtt{MEM}}$) contains a Memory Capability ($\mathtt{CAP}_{\mathtt{MEM}}$) that refers to a page frame in physical memory.

2.1.3 I/O Port Space

1. Each empty slot of the I/O Port Space ($\mathtt{SPC}_{\mathtt{PIO}}$) contains a Null Capability ($\mathtt{CAP}_{\emptyset}$).
2. Each non-empty slot of the I/O Port Space ($\mathtt{SPC}_{\mathtt{PIO}}$) contains a I/O Port Capability ($\mathtt{CAP}_{\mathtt{PIO}}$) that refers to an I/O port.

2.1.4 MSR Space

1. Each empty slot of the MSR Space ($\mathtt{SPC}_{\mathtt{MSR}}$) contains a Null Capability ($\mathtt{CAP}_{\emptyset}$).
2. Each non-empty slot of the MSR Space ($\mathtt{SPC}_{\mathtt{MSR}}$) contains a MSR Capability ($\mathtt{CAP}_{\mathtt{MSR}}$) that refers to an MSR.

2.2 Execution Context

1. The Execution Context (EC) is an abstraction for an activity within a PD.
2. Access to an Execution Context (EC) is controlled by an EC Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{EC}}}$).
3. An EC is permanently bound to the PD in which it was created.
4. An EC may optionally have an SC bound to it.
5. There exist two flavors of execution context:
   - Threads
   - Virtual CPUs
6. An EC comprises the following state:

- Reference to PD (2.1)
- Event Selector Base (SEL_{EVT})
- User Thread Control Block (UTCB) (3.3)
- CPU Number (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. Access to a Scheduling Context (SC) is controlled by an SC Object Capability ($CAP_{OBJ_{SC}}$).
3. An SC is permanently bound to exactly one physical CPU.
4. At any point in time, an SC is bound to exactly one EC.
5. Donation of an SC to another EC temporarily binds the SC to that other EC.
6. A scheduling context comprises the following state:
   - Reference to EC (2.2)
   - Time quantum
   - Priority

2.4 Portal

1. A Portal (PT) represents a dedicated entry point into the PD in which the portal was created.
2. Access to a Portal (PT) is controlled by a PT Object Capability ($CAP_{OBJ_{PT}}$).
3. A PT is permanently bound to exactly one EC.
4. A portal comprises the following state:
   - Reference to EC (2.2)
   - Message Transfer Descriptor (MTD) (3.4)
   - Entry instruction pointer
   - Portal Identifier (PID)

2.5 Semaphore

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

Part III

Application Programming Interface

3 Data Types

3.1 Capability

A Capability (CAP) is a reference to a resource plus associated auxiliary data, such as access permissions.

Capabilities are opaque and immutable for applications – they cannot be inspected or modified directly; instead applications refer to a Capability via a Capability Selector (SEL).

3.1.1 Null Capability

A Null Capability ($\mathtt{CAP}_{\emptyset}$) does not refer to anything and carries no permissions.

3.1.2 Object Capability

An Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}}$) is stored in the Object Space ($\mathtt{SPC}_{\mathtt{OBJ}}$) of a PD and refers to a kernel object.

3.1.2.1 PD Object Capability

A PD Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{PD}}}$) refers to a Protection Domain (PD) and carries the following permissions:

4	3	2	1	0
ASSIGN	SC	EC PT SM	PD	CTRL

CTRL	ctrl_pd permitted if set.
PD	create_pd permitted if set.
EC PT SM	create_ec, create_pt, create_sm permitted it set.
SC	create_sc permitted if set.
ASSIGN	assign_dev permitted if set.

3.1.2.2 EC Object Capability

An EC Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{EC}}}$) refers to an Execution Context (EC) and carries the following permissions:

4	3	2	1	0
-	$\mathtt{BIND}_{\mathtt{SC}}$	$\mathtt{BIND}_{\mathtt{PT}}$	-	CTRL

CTRL	ctrl_ec permitted if set.
$\mathtt{BIND}_{\mathtt{PT}}$	create_pt can bind a Portal (PT) to the EC if set.
$\mathtt{BIND}_{\mathtt{SC}}$	create_sc can bind a Scheduling Context (SC) to the EC if set.

3.1.2.3 SC Object Capability

An SC Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{SC}}}$) refers to a Scheduling Context (SC) and carries the following permissions:

4	3	2	1	0
-	-	-	-	CTRL

CTRL	ctrl_sc permitted if set.

3.1.2.4 PT Object Capability

A PT Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{PT}}}$) refers to a Portal (PT) and carries the following permissions:

4	3	2	1	0
-	-	EVENT	CALL	CTRL

CTRL	`ctrl_pt` permitted if set.
CALL	`ipc_call` permitted if set.
EVENT	Delivery of events permitted if set.

3.1.2.5 SM Object Capability

An SM Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{SM}}}$) refers to a Semaphore (SM) and carries the following permissions:

4	3	2	1	0
ASSIGN	-	-	$\mathtt{CTRL}_{\mathtt{DN}}$	$\mathtt{CTRL}_{\mathtt{UP}}$

$\mathtt{CTRL}_{\mathtt{UP}}$	`ctrl_sm` (Up) permitted if set.
$\mathtt{CTRL}_{\mathtt{DN}}$	`ctrl_sm` (Down) permitted if set.
ASSIGN	`assign_int` permitted if set.

3.1.3 Memory Capability

A Memory Capability ($\mathtt{CAP}_{\mathtt{MEM}}$) is stored in the Memory Space ($\mathtt{SPC}_{\mathtt{MEM}}$) of a PD, refers to a 4KB page frame, and carries the following permissions:

4	3	2	1	0
-	$\mathtt{X}_{\mathtt{S}}$	$\mathtt{X}_{\mathtt{U}}$	W	R

R	the memory page is readable if set.
W	the memory page is writable if set.
$\mathtt{X}_{\mathtt{U}}$	the memory page is executable (in user mode) if set. †
$\mathtt{X}_{\mathtt{S}}$	the memory page is executable (in supervisor mode) if set. †

3.1.4 I/O Port Capability

A I/O Port Capability ($\mathtt{CAP}_{\mathtt{PIO}}$) is stored in the I/O Port Space ($\mathtt{SPC}_{\mathtt{PIO}}$) of a PD, refers to an I/O port, and carries the following permissions:

4	3	2	1	0
-	-	-	-	A

A	the I/O port is accessible if set.

3.1.5 MSR Capability

A MSR Capability ($\mathtt{CAP}_{\mathtt{MSR}}$) is stored in the MSR Space ($\mathtt{SPC}_{\mathtt{MSR}}$) of a PD, refers to a Model-Specific Register (MSR), and carries the following permissions:

4	3	2	1	0
-	-	-	-	A

A	the MSR is accessible if set.

†If the hardware supports only combined execute permissions (X) for both modes, then $\mathtt{X} = \mathtt{X}_{\mathtt{U}} \vee \mathtt{X}_{\mathtt{S}}$.

3.2 Capability Selector

A Capability Selector (SEL) is a user-visible unsigned number as follows:

- An Object Capability Selector (SEL_{OBJ}) serves as an index into the Object Space (SPC_{OBJ}) of a Protection Domain (PD) and selects a slot that either contains an Object Capability (CAP_{OBJ}) or a Null Capability ($CAP_{\emptyset}$).
- A Memory Capability Selector (SEL_{MEM}) serves as an index into the Memory Space (SPC_{MEM}) of a Protection Domain (PD) and selects a slot that either contains a Memory Capability (CAP_{MEM}) or a Null Capability ($CAP_{\emptyset}$).
- A I/O Port Capability Selector (SEL_{PIO}) serves as an index into the I/O Port Space (SPC_{PIO}) of a Protection Domain (PD) and selects a slot that either contains a I/O Port Capability (CAP_{PIO}) or a Null Capability ($CAP_{\emptyset}$).
- An MSR Capability Selector (SEL_{MSR}) serves as an index into the MSR Space (SPC_{MSR}) of a Protection Domain (PD) and selects a slot that either contains an MSR Capability (CAP_{MSR}) or a Null Capability ($CAP_{\emptyset}$).

3.3 User Thread Control Block

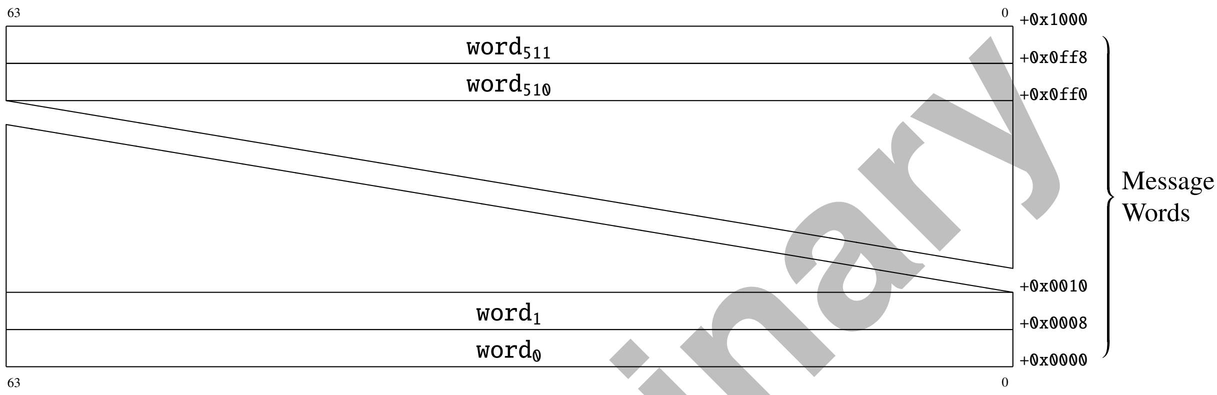
Each host EC (local/global thread) has its own User Thread Control Block (UTCB), which is mapped into the Memory Space (SPC_{MEM}) of the PD in which that EC is executing. A guest EC (virtual CPU) does not have a UTCB.

A User Thread Control Block (UTCB) has a size of one page (4096 bytes).

To ensure proper visibility of loads and stores with relaxed memory ordering, application programs are expected to access a UTCB only from the EC to which that UTCB is bound.

3.3.1 Regular Layout

During regular IPC (see 3.4.1), the UTCB is used for data transfer and has a regular layout with 512 message words.



The data transfer from one UTCB to another UTCB is defined as follows:

- The data transfer is performed by the CPU on which the caller EC and callee EC execute.
- The data transfer uses the regular layout.
- The data is copied from low words to high words, beginning with $\mathtt{word}_0$.
- The granularity of the loads and stores used for copying is **undefined**.
- Loads from and stores to the UTCB are **non-atomic** and use **relaxed** memory ordering.

3.3.2 Architectural Layout

During architectural IPC (see 3.4.2), the UTCB is used for state transfer and has an architectural layout (ARM, x86).

The state transfer between the architectural registers and a UTCB is defined as follows:

- The state transfer is performed by the CPU on which the faulting EC and callee EC execute.
- The state transfer uses the architectural layout.
- The state is copied between architectural registers and the UTCB in an **undefined** order.
- The granularity of the loads and stores used for copying is **undefined**.
- Loads from and stores to the UTCB are **non-atomic** and use **relaxed** memory ordering.

3.4 Message Transfer Descriptor

3.4.1 Regular IPC

For regular Inter-Process Communication (`IPC`), the Message Transfer Descriptor (`MTD`) is provided by the sender, passed to the receiver, and uses the following layout:

31 – 9	8 – 0
-	`UTCB Message Words - 1`

The `MTD` controls the data transfer (see 3.3.1) as shown in Figure 3.1:

- During `ipc_call`, it specifies the number of message words to transfer from the UTCB of the caller `EC` (sender) to the UTCB of the callee `EC` (receiver).
- During `ipc_reply`, it specifies the number of message words to transfer from the UTCB of the callee `EC` (sender) to the UTCB of the caller `EC` (receiver).

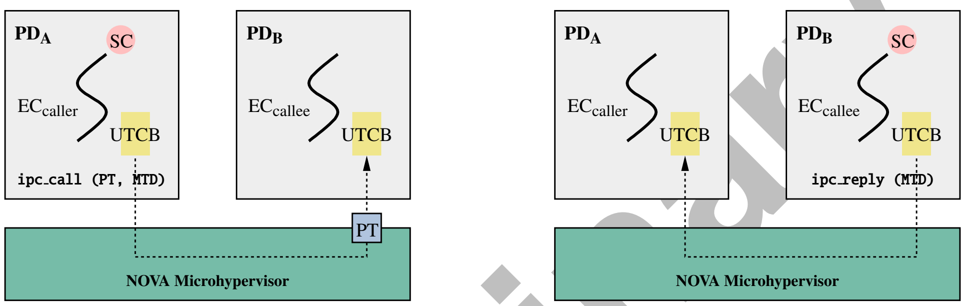


Figure 3.1: Regular `IPC`

3.4.2 Architectural IPC

For exceptions and intercepts, the Message Transfer Descriptor (`MTD`) is provided by the architectural event-specific portal (ARM, x86) or sender, passed to the receiver, and uses an architectural bitfield layout (ARM, x86):

- If a bit is `0`, then the microhypervisor does **not** transmit the architectural state associated with that bit.
- If a bit is `1`, then the microhypervisor transmits the architectural state associated with that bit.

The `MTD` controls the state transfer (see 3.3.2) as shown in Figure 3.2:

- During an exception/intercept, it specifies the subset of registers to transfer from the architectural state of the faulting `EC` (sender) to the UTCB of the callee `EC` (receiver).
- During `ipc_reply`, it specifies the subset of registers to transfer from the UTCB of the callee `EC` (sender) to the architectural state of the faulting `EC` (receiver).

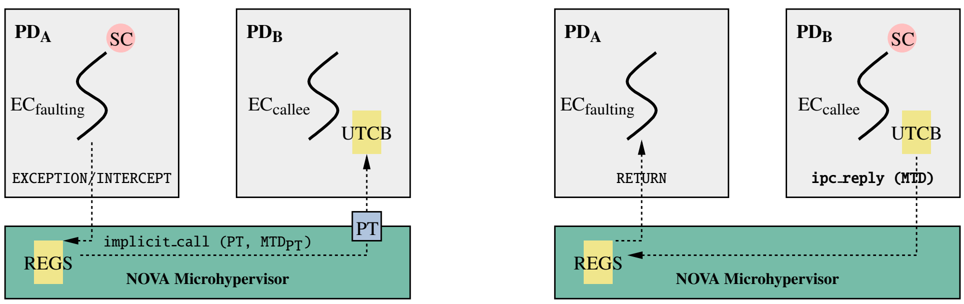


Figure 3.2: Architectural `IPC`

4 Hypercalls

4.1 Definitions

4.1.1 Hypercall Numbers

Each hypercall is identified by a unique number. The following hypercalls are currently defined:

Number	Hypercall	Section
`0x0`	`ipc_call`	4.2.1
`0x1`	`ipc_reply`	4.2.2
`0x2`	`create_pd`	4.3.1
`0x3`	`create_ec`	4.3.2
`0x4`	`create_sc`	4.3.3
`0x5`	`create_pt`	4.3.4
`0x6`	`create_sm`	4.3.5
`0x7`	`ctrl_pd`	4.4.1
`0x8`	`ctrl_ec`	4.4.2
`0x9`	`ctrl_sc`	4.4.3
`0xa`	`ctrl_pt`	4.4.4
`0xb`	`ctrl_sm`	4.4.5
`0xc`	`ctrl_hw`	4.4.6
`0xd`	`assign_int`	4.5.1
`0xe`	`assign_dev`	4.5.2
`0xf`	*reserved for future use*	

4.1.2 Status Codes

Hypercalls return a status code to indicate success or failure. The following status codes are currently defined:

Number	Status Code	Description
`0x0`	`SUCCESS`	Operation Successful
`0x1`	`TIMEOUT`	Operation Timeout
`0x2`	`ABORTED`	Operation Abort
`0x3`	`OVRFLOW`	Operation Overflow
`0x4`	`BAD_HYP`	Invalid Hypercall
`0x5`	`BAD_CAP`	Invalid Capability
`0x6`	`BAD_PAR`	Invalid Parameter
`0x7`	`BAD_FTR`	Invalid Feature
`0x8`	`BAD_CPU`	Invalid CPU Number
`0x9`	`BAD_DEV`	Invalid Device ID
`0xa`	`INS_MEM`	Insufficient Memory

4.1.3 Space Type

Number	$\mathrm{TYPE}_{\mathrm{SPC}}$	Contains	Indexed By	Description
`0x0`	$\mathrm{SPC}_{\mathrm{OBJ}}$	$\mathrm{CAP}_{\mathrm{OBJ}}$	$\mathrm{SEL}_{\mathrm{OBJ}}$	Object Space
`0x1`	$\mathrm{SPC}_{\mathrm{MEM}}$	$\mathrm{CAP}_{\mathrm{MEM}}$	$\mathrm{SEL}_{\mathrm{MEM}}$	Memory Space
`0x2`	$\mathrm{SPC}_{\mathrm{PIO}}$	$\mathrm{CAP}_{\mathrm{PIO}}$	$\mathrm{SEL}_{\mathrm{PIO}}$	I/O Port Space
`0x3`	$\mathrm{SPC}_{\mathrm{MSR}}$	$\mathrm{CAP}_{\mathrm{MSR}}$	$\mathrm{SEL}_{\mathrm{MSR}}$	MSR Space

4.1.4 Table Type

Number	TYPE_{TBL}	Description
0x0	CPU_HST	CPU Page Table for Host Accesses
0x1	CPU_GST	CPU Page Table for Guest Accesses
0x2	DMA_HST	DMA Page Table for Host Accesses
0x3	DMA_GST	DMA Page Table for Guest Accesses

4.2 Communication

4.2.1 IPC Call

Parameters:

```
status = ipc_call (SEL_OBJ pt,       // Portal
                   MTD     &mtd);    // Message Transfer Descriptor
```

Flags:

0	0	0	T
3	2	1	0

Description:

Sends a message from $EC_{CURRENT}$ (caller) to the EC (callee) to which the specified Portal (PT) is bound.

Prior to the hypercall:

- { $PD_{CURRENT}$, SEL_{OBJ} pt } must refer to a PT Object Capability ($CAP_{OBJ_{PT}}$) with permission CALL.

If the hypercall completed successfully:

- If **T=0 (No Timeout)**: If the callee EC was busy handling another request, then the caller EC has helped run that request to completion, i.e. until the callee EC became available again.
- The microhypervisor has transferred a message from the UTCB of the caller EC to the UTCB of the callee EC. The content of that message is defined by the MTD mtd, which has been passed from the caller EC to the callee EC.
- The hypercall returns once the callee EC has issued an ipc_reply. Upon return, the UTCB of the caller EC and the parameter mtd have been updated by the reply message.
- The Current Scheduling Context ($SC_{CURRENT}$) has been donated to the callee EC upon ipc_call and returned back upon ipc_reply, thereby accounting the entire handling of the request to $SC_{CURRENT}$.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $PD_{CURRENT}$, SEL_{OBJ} pt } did not refer to a PT Object Capability ($CAP_{OBJ_{PT}}$) or that capability had insufficient permissions.

BAD_CPU

- Caller EC and callee EC are on different CPUs.

TIMEOUT

- The callee EC is busy handling another request – only if **T=1 (Timeout)**.

ABORTED

- The callee EC is dead and the operation aborted.

4.2.2 IPC Reply

Parameters:

```
pid = ipc_reply (MTD &mtd);                    // Message Transfer Descriptor
```

Flags:

0	0	0	0
3	2	1	0

Description:

Sends a reply message from $\text{EC}_{\text{CURRENT}}$ (callee) back to the caller EC (if one exists) and subsequently waits for the next incoming message.

If the hypercall completed successfully:

- If a caller EC exists:
  - The microhypervisor has transferred a reply message from the UTCB of the callee EC back to the UTCB of the caller EC.
  - The content of that reply message is defined by the MTD `mtd`, which has been passed from the callee EC back to the caller EC.
  - The Current Scheduling Context ($\text{SC}_{\text{CURRENT}}$) that had been donated to the callee EC upon `ipc_call` has been returned back to the caller EC.
- $\text{EC}_{\text{CURRENT}}$ blocks until the next incoming message arrives on any Portal (PT) bound to it.

Status:

This hypercall does not return directly.

Instead, when the next message arrives via a subsequent `ipc_call` to any Portal (PT) bound to the callee EC:

- The microhypervisor passes the Portal Identifier (PID) of the called PT to the callee EC.
- The UTCB of the callee EC and the parameter `mtd` have been updated by the incoming message.
- Execution of the callee EC continues at the Instruction Pointer (IP) configured in the called PT.

4.3 Object Creation

4.3.1 Create Protection Domain

Parameters:

```
status = create_pd (SEL_OBJ sel,         // Created PD
                    SEL_OBJ own);        // Owner PD
```

Flags:

∅	∅	∅	∅
3	2	1	0

Description:

Creates a new Protection Domain (PD).

Prior to the hypercall:

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own } must refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) with permission PD.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sel } must refer to a Null Capability ($\mathrm{CAP}_{\emptyset}$).

If the hypercall completed successfully:

- A new Protection Domain (PD) has been created.
- The resources for the created PD were accounted to the PD referred to by { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own }.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sel } refers to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) for the created PD.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own } did not refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) or that capability had insufficient permissions.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sel } did not refer to a Null Capability ($\mathrm{CAP}_{\emptyset}$).

INS_MEM

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own } had insufficient memory resources for PD creation.

4.3.2 Create Execution Context

Parameters:

```
status = create_ec (SEL_OBJ  sel,      // Created EC
                    SEL_OBJ  own,      // Owner PD
                    SEL_MEM  utcb,     // UTCB Address (Page Number)
                    UINT     cpu,      // CPU Number
                    UINT     sp,       // Initial Stack Pointer
                    SEL_EVT  evt);     // Event Selector Base
```

Flags:

0	F	V	G
3	2	1	0

Description:

Creates a new Execution Context (EC).

Prior to the hypercall:

- { $PD_{CURRENT}$, SEL_{OBJ} own } must refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) with permission EC.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } must refer to a Null Capability ($CAP_{\emptyset}$).

If the hypercall completed successfully:

- If **V=0,G=0 (Local Thread)**: A new host Execution Context (EC) has been created with its UTCB mapped at virtual page number utcb and its initial Stack Pointer (SP) set to sp. Portals (PTs) can subsequently be bound to that EC and the EC will run whenever any of those bound portals is called.
- If **V=0,G=1 (Global Thread)**: A new host Execution Context (EC) has been created with its UTCB mapped at virtual page number utcb and its initial Stack Pointer (SP) set to sp. The EC will generate a startup exception the first time a Scheduling Context (SC) is bound to it.
- If **V=1 (Virtual CPU)**: A new guest Execution Context (EC) has been created. The EC will generate a startup exception the first time a Scheduling Context (SC) is bound to it. The parameters utcb, sp and the G-flag were ignored.
- The created EC will be able to use FPU instructions only if the F-flag is set. Otherwise any FPU access by that EC will generate an exception.
- The created EC is bound to the PD referred to by { $PD_{CURRENT}$, SEL_{OBJ} own } on CPU cpu with its Event Selector Base (SEL_{EVT}) set to evt.
- The resources for the created EC were accounted to the PD referred to by { $PD_{CURRENT}$, SEL_{OBJ} own }.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } refers to an EC Object Capability ($CAP_{OBJ_{EC}}$) for the created EC.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $PD_{CURRENT}$, SEL_{OBJ} own } did not refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) or that capability had insufficient permissions.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } did not refer to a Null Capability ($CAP_{\emptyset}$).

BAD_CPU

- The CPU number is invalid.

BAD_FTR

- Virtual CPUs are not supported on the machine.

BAD_PAR

- UTCB region is not free or outside the user-accessible memory range.

INS_MEM

- { $PD_{CURRENT}$, SEL_{OBJ} own } had insufficient memory resources for EC creation.

4.3.3 Create Scheduling Context

Parameters:

```
status = create_sc (SEL_OBJ sel,        // Created SC
                    SEL_OBJ own,        // Owner PD
                    SEL_OBJ ec,         // Bound EC
                    UINT    quantum,    // Scheduling Time Quantum
                    UINT    prio);      // Scheduling Priority
```

Flags:

3	2	1	0
0	0	0	0

Description:

Creates a new Scheduling Context (SC).

Prior to the hypercall:

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own } must refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) with permission SC.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ ec } must refer to an EC Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{EC}}}$) with permission $\mathtt{BIND}_{\mathrm{SC}}$.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sel } must refer to a Null Capability ($\mathrm{CAP}_{\emptyset}$).

If the hypercall completed successfully:

- A new Scheduling Context (SC) has been created.
- The created SC is bound to the EC referred to by { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ ec } on the CPU of that EC with its scheduling parameters set to quantum and priority.
- The resources for the created SC were accounted to the PD referred to by { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own }.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sel } refers to an SC Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{SC}}}$) for the created SC.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own } did not refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) or that capability had insufficient permissions.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ ec } did not refer to a EC Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{EC}}}$) or that capability had insufficient permissions.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sel } did not refer to a Null Capability ($\mathrm{CAP}_{\emptyset}$).
- Binding the SC to the EC failed, e.g. because the EC is a local EC.

BAD_PAR

- Time quantum or priority was zero.

INS_MEM

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own } had insufficient memory resources for SC creation.

4.3.4 Create Portal

Parameters:

```
status = create_pt (SEL_OBJ  sel,        // Created PT
                    SEL_OBJ  own,        // Owner PD
                    SEL_OBJ  ec,         // Bound EC
                    UINT     ip);        // Instruction Pointer
```

Flags:

0	0	0	0
3	2	1	0

Description:

Creates a new Portal (PT).

Prior to the hypercall:

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `own` } must refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) with permission PT.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `ec` } must refer to an EC Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{EC}}}$) with permission $\mathrm{BIND}_{\mathrm{PT}}$.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `sel` } must refer to a Null Capability ($\mathrm{CAP}_{\emptyset}$).

If the hypercall completed successfully:

- A new Portal (PT) has been created.
- The created PT is bound to the EC referred to by { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `ec` } on the CPU of that EC, with its portal Instruction Pointer (IP) set to `ip`, its initial MTD set to 0 and its initial PID set to 0.
- The resources for the created PT were accounted to the PD referred to by { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `own` }.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `sel` } refers to an PT Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PT}}}$) for the created PT.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `own` } did not refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) or that capability had insufficient permissions.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `ec` } did not refer to a EC Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{EC}}}$) or that capability had insufficient permissions.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `sel` } did not refer to a Null Capability ($\mathrm{CAP}_{\emptyset}$).
- Binding the PT to the EC failed, e.g. because the EC is not a local EC.

INS_MEM

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `own` } had insufficient memory resources for PT creation.

4.3.5 Create Semaphore

Parameters:

```
status = create_sm (SEL_OBJ sel,       // Created SM
                    SEL_OBJ own,       // Owner PD
                    UINT    cnt);      // Initial Counter Value
```

Flags:

0	0	0	0
3	2	1	0

Description:

Creates a new Semaphore (SM).

Prior to the hypercall:

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own } must refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) with permission SM.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sel } must refer to a Null Capability ($\mathrm{CAP}_{\emptyset}$).

If the hypercall completed successfully:

- A new Semaphore (SM) has been created.
- The created SM has its initial counter value set to cnt.
- The resources for the created SM were accounted to the PD referred to by { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own }.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sel } refers to an SM Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{SM}}}$) for the created SM.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own } did not refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) or that capability had insufficient permissions.
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sel } did not refer to a Null Capability ($\mathrm{CAP}_{\emptyset}$).

INS_MEM

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ own } had insufficient memory resources for SM creation.

4.4 Object Control

4.4.1 Control Protection Domain

Parameters:

```
status = ctrl_pd ( SEL_OBJ  spd,      // Protection Domain: Source
                   SEL_OBJ  dpd,      // Protection Domain: Destination
                   SEL      src,      // Base Selector: Source
                   SEL      dst,      // Base Selector: Destination
                   UINT     ord,      // Order
                   UINT     pmm,      // Permission Mask
                   TYPE_SPC spc,      // Space Type
                   TYPE_TBL tbl,      // Table Type
                   ATTR_CA  ca,       // Cacheability Attribute
                   ATTR_SH  sh);      // Shareability Attribute
```

Flags:

0	0	0	0
3	2	1	0

Description:

Takes capabilities from the Source Protection Domain (PD) and grants them to the Destination Protection Domain (PD) and thereby optionally reduces the permissions of the destination capabilities.

Prior to the hypercall:

- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} spd } must refer to a PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) with permission CTRL.
- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} dpd } must refer to a PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) with permission CTRL.
- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} dpd } must not refer to a PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) for PD_{NOVA}.
- SEL src and SEL dst must be order-aligned, i.e. src≡0 (mod 2^{ord}) and dst≡0 (mod 2^{ord}).
- TYPE_{SPC} spc and TYPE_{TBL} tbl must be valid, i.e. supported by the architecture.
- ATTR_{CA} ca and ATTR_{SH} sh must be valid, i.e. supported by the architecture.

If the hypercall completed successfully:

- If **spc=SPC_{OBJ}**: All CAP_{OBJ} and $\text{CAP}_{\emptyset}$ from source SEL range { PD spd, SEL_{OBJ} src...src+2^{ord}-1 } were delegated to destination SEL range { PD dpd, SEL_{OBJ} dst...dst+2^{ord}-1 }. Any pre-existing CAP_{OBJ} in the destination selector range were revoked. The parameters tbl, ca and sh were ignored.
- If **spc=SPC_{MEM}**: All CAP_{MEM} and $\text{CAP}_{\emptyset}$ from source SEL range { PD spd, SEL_{MEM} src...src+2^{ord}-1 } were delegated to destination SEL range { PD dpd, SEL_{MEM} dst...dst+2^{ord}-1 }. Any pre-existing CAP_{MEM} in the destination selector range were revoked.
- If **spc=SPC_{PIO}**: All CAP_{PIO} and $\text{CAP}_{\emptyset}$ from source SEL range { PD spd, SEL_{PIO} src...src+2^{ord}-1 } were delegated to destination SEL range { PD dpd, SEL_{PIO} dst...dst+2^{ord}-1 }. Any pre-existing CAP_{PIO} in the destination selector range were revoked. The parameters tbl, ca and sh were ignored.
- If **spc=SPC_{MSR}**: All CAP_{MSR} and $\text{CAP}_{\emptyset}$ from source SEL range { PD spd, SEL_{MSR} src...src+2^{ord}-1 } were delegated to destination SEL range { PD dpd, SEL_{MSR} dst...dst+2^{ord}-1 }. Any pre-existing CAP_{MSR} in the destination selector range were revoked. The parameters tbl, ca and sh were ignored.
- The permissions of each destination capability were masked by computing the logical AND of the permissions of the respective source capability and the permission mask pmm, i.e.
  - for bits set (1) in pmm, the respective permissions were *inherited* from the source capability.
  - for bits clear (0) in pmm, the respective permissions were *removed* for the destination capability.
- If the source capability was a Null Capability ($\text{CAP}_{\emptyset}$) or if the destination capability would have had zero permissions after masking, then the destination capability is now a Null Capability ($\text{CAP}_{\emptyset}$).
- The resources for storing the granted capabilities were accounted to the PD referred to by { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} dpd }.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $\mathtt{PD}_{\mathtt{CURRENT}}$, $\mathtt{SEL}_{\mathtt{OBJ}}$ `spd` } did not refer to a PD Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{PD}}}$) or that capability had insufficient permissions.
- { $\mathtt{PD}_{\mathtt{CURRENT}}$, $\mathtt{SEL}_{\mathtt{OBJ}}$ `dpd` } did not refer to a PD Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{PD}}}$) or that capability had insufficient permissions.
- { $\mathtt{PD}_{\mathtt{CURRENT}}$, $\mathtt{SEL}_{\mathtt{OBJ}}$ `dpd` } referred to a PD Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{PD}}}$) for $\mathtt{PD}_{\mathtt{NOVA}}$.

BAD_PAR

- SEL `src` or SEL `dst` was not order-aligned.
- SEL $\mathtt{src}+2^{\mathtt{ord}}-1$ or SEL $\mathtt{dst}+2^{\mathtt{ord}}-1$ was larger than the maximum selector number.
- If **spc=$\mathtt{SPC}_{\mathtt{PIO}}$ or spc=$\mathtt{SPC}_{\mathtt{MSR}}$**: SEL `src` was not equal to SEL `dst`.
- $\mathtt{TYPE}_{\mathtt{SPC}}$ `spc` or $\mathtt{TYPE}_{\mathtt{TBL}}$ `tbl` was not valid, i.e. not supported by the architecture.
- $\mathtt{ATTR}_{\mathtt{CA}}$ `ca` or $\mathtt{ATTR}_{\mathtt{SH}}$ `sh` was not valid, i.e. not supported by the architecture.

INS_MEM †

- { $\mathtt{PD}_{\mathtt{CURRENT}}$, $\mathtt{SEL}_{\mathtt{OBJ}}$ `dpd` } had insufficient memory resources for allocating the storage required for granting all destination capabilities. This constitutes a partial failure of the operation, because those destination capabilities, for which storage allocation succeeded or storage already existed, have been granted.

†Planned, but currently not implemented. May change during a future implementation.

4.4.2 Control Execution Context

Parameters:

```
status = ctrl_ec (SEL_OBJ ec);            // Execution Context
```

Flags:

0	0	0	S
3	2	1	0

Description:

Prior to the hypercall:

- { $PD_{CURRENT}$, SEL_{OBJ} ec } must refer to a EC Object Capability ($CAP_{OBJ_{EC}}$) with permission CTRL.

If the hypercall completed successfully:

- The EC referred to by { $PD_{CURRENT}$, SEL_{OBJ} ec } has been forced to enter the microhypervisor. It will generate a recall exception prior to its next exit from the microhypervisor and will traverse through the respective Portal (PT).
- If **S=0 (Weak)**: the hypercall returns as soon as the recall exception has been *pended*, i.e. the EC may not have entered the microhypervisor yet.
- If **S=1 (Strong)**: the hypercall returns as soon as the recall exception has been *observed*, i.e the EC will have entered the microhypervisor.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $PD_{CURRENT}$, SEL_{OBJ} ec } did not refer to a EC Object Capability ($CAP_{OBJ_{EC}}$) or that capability had insufficient permissions.

4.4.3 Control Scheduling Context

Parameters:

```
status = ctrl_sc (SEL_OBJ  sc,          // Scheduling Context
                  UINT     &ticks);     // Total Consumed Execution Time
```

Flags:

3	2	1	0
0	0	0	0

Description:

Prior to the hypercall:

- { $PD_{CURRENT}$, SEL_{OBJ} sc } must refer to an SC Object Capability ($CAP_{OBJ_{SC}}$) with permission CTRL.

If the hypercall completed successfully:

- The microhypervisor has returned the total consumed execution time in `ticks` for the SC referred to by { $PD_{CURRENT}$, SEL_{OBJ} sc }.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $PD_{CURRENT}$, SEL_{OBJ} sc } did not refer to an SC Object Capability ($CAP_{OBJ_{SC}}$) or that capability had insufficient permissions.

4.4.4 Control Portal

Parameters:

```
status = ctrl_pt (SEL_OBJ pt,          // Portal
                  UINT    pid,         // Portal Identifier
                  MTD     mtd);        // Message Transfer Descriptor
```

Flags:

3	2	1	0
0	0	0	0

Description:

Prior to the hypercall:

- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} pt } must refer to a PT Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PT}}}$) with permission CTRL.

If the hypercall completed successfully:

- The microhypervisor has set the Portal Identifier (PID) to pid and the Message Transfer Descriptor (MTD) to mtd for the Portal referred to by { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} pt }.
- Subsequent portal traversals will use the new MTD and return the new PID.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} pt } did not refer to a PT Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PT}}}$) or that capability had insufficient permissions.

4.4.5 Control Semaphore

Parameters:

```
status = ctrl_sm (SEL_OBJ sm,               // Semaphore
                  UINT   ticks);            // Deadline Timeout
```

Flags:

3	2	1	0
0	0	Z	D

Description:

Prior to the hypercall:

- If D=0 **(Up)**: { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sm } must refer to a SM Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{SM}}}$) with permission $\mathrm{CTRL}_{\mathrm{UP}}$.
- If D=1 **(Down)**: { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sm } must refer to a SM Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{SM}}}$) with permission $\mathrm{CTRL}_{\mathrm{DN}}$.

If the hypercall completed successfully:

- If D=0 **(Up)**: if there were ECs blocked on the semaphore, then the microhypervisor has released one of those blocked ECs. Otherwise, the microhypervisor has incremented the semaphore counter. The deadline timeout value and the Z-flag were ignored.
- If D=1 **(Down)**: if the semaphore counter was larger than zero, then the microhypervisor has decremented the semaphore counter (Z=0) or set it to zero (Z=1). Otherwise, the microhypervisor has blocked $\mathrm{EC}_{\mathrm{CURRENT}}$ on the semaphore. If the deadline timeout value was non-zero, $\mathrm{EC}_{\mathrm{CURRENT}}$ unblocks with a timeout status when the architectural timer reaches or exceeds the specified ticks value.

Blocking and releasing of ECs on a semaphore uses the FIFO queueing discipline.

Status:

SUCCESS

- The hypercall completed successfully.

TIMEOUT

- If D=1: Down operation aborted when the timeout triggered.

OVRFLOW

- If D=0: Up operation aborted because the semaphore counter would overflow.

BAD_CAP

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ sm } did not refer to a SM Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{SM}}}$) or that capability had insufficient permissions.

BAD_CPU

- If D=1 on an interrupt semaphore: Attempt to wait for the interrupt on a different CPU than the CPU to which that interrupt has been routed via `assign_int`.

4.4.6 Control Hardware

Parameters:

```
status = ctrl_hw (UINT &arg0,          // Parameter 0
                  UINT &arg1,          // Parameter 1
                  UINT &arg2,          // Parameter 2
                  UINT &arg3,          // Parameter 3
                  UINT  arg4,          // Parameter 4
                  UINT  arg5,          // Parameter 5
                  UINT  arg6);         // Parameter 6
```

Flags:

1	1	1	1
3	2	1	0

Description:

Performs a firmware call via SMC.

Prior to the hypercall:

- $PD_{CURRENT}$ must be the Root Protection Domain (PD_{ROOT}).
- Flags must be set to 0b1111 to indicate a firmware call.
- The SMC number must be passed in arg0 and must represent an atomic SIP SMC.
- The SMC parameters must be passed in arg1 ... arg6.

If the hypercall completed successfully:

- The SMC return values will be passed in arg0 ... arg3.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_HYP

- The hypercall was not issued from the Root Protection Domain (PD_{ROOT}).

BAD_PAR

- The flags value was not 0b1111 or the SMC did not represent an atomic SIP call.

BAD_FTR

- The CPU does not support SMCs.

4.5 Interrupt and Device Assignment

4.5.1 Assign Interrupt

Parameters:

```
status = assign_int (SEL_OBJ sm,           // Interrupt Semaphore
                     UINT  cpu,            // CPU Number
                     UINT  dev,            // MSI Authorized Device
                     UINT &msi_addr,       // MSI Message Address
                     UINT &msi_data);      // MSI Message Data
```

Flags:

G	P	T	M
3	2	1	0

Description:

Configures an interrupt and routes it to the specified CPU.

Prior to the hypercall:

- { $PD_{CURRENT}$, SEL_{OBJ} sm } must refer to a SM Object Capability ($CAP_{OBJ_{SM}}$) with permission ASSIGN.
- $CAP_{OBJ_{SM}}$ must refer to an interrupt semaphore and thereby identifies the interrupt.

If the hypercall completed successfully:

- The interrupt referred to by { $PD_{CURRENT}$, SEL_{OBJ} sm } has been routed to the CPU cpu.
- Mask
  - **M=0**: The interrupt is now unmasked, i.e. it will be signaled on the semaphore.
  - **M=1**: The interrupt is now masked, i.e. it will not be signaled on the semaphore.
- Trigger
  - **T=0**: The interrupt is now configured for edge-triggered operation.
  - **T=1**: The interrupt is now configured for level-triggered operation.
- Polarity
  - **P=0**: The interrupt is now configured for active-high operation.
  - **P=1**: The interrupt is now configured for active-low operation.
- Guest
  - **G=0**: The interrupt is now host-owned.
  - **G=1**: The interrupt is now guest-owned (VM pass-through).
- If the interrupt is an MSI, only the PCI device referred to by dev will be authorized to generate that MSI. The device driver must program the returned msi_addr and msi_data values into the MSI registers of that device to ensure proper interrupt operation. If the interrupt is pin-based, the parameter dev was ignored and the parameters msi_addr and msi_data return 0.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CPU

- The specified CPU number was invalid.

BAD_CAP

- { $PD_{CURRENT}$, SEL_{OBJ} sm } did not refer to a SM Object Capability ($CAP_{OBJ_{SM}}$) or that capability had insufficient permissions.
- $CAP_{OBJ_{SM}}$ did not refer to an interrupt semaphore.

4.5.2 Assign Device

Parameters:

```
status = assign_dev (SEL_OBJ  pd,        // Protection Domain
                     SEL_MEM  smmu,      // SMMU Address (Page Number)
                     UINT     dev,       // Assigned Device (SID/BDF)
                     TYPE_TBL tbl);      // Table Type
```

Flags:

0	0	0	0
3	2	1	0

Description:

Assigns the specified device (*) to the specified Protection Domain (PD):

- **ARM**: dev encodes the SID of the device and also the SMMU resources (stream mapping group, translation context) to be used for managing that device.
- **x86**: dev encodes the BDF of the device. There are no SMMU resources needed.

Prior to the hypercall:

- $PD_{CURRENT}$ must be the Root Protection Domain (PD_{ROOT}).
- { $PD_{CURRENT}$, SEL_{OBJ} pd } must refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) with permission ASSIGN.
- { PD_{NOVA}, SEL_{MEM} smmu } must refer to the physical address of an SMMU device.
- The SID/BDF and SMMU resources encoded in dev must be supported by the hardware (see 6.5.1).
- $TYPE_{TBL}$ tbl must refer to a DMA page table.

If the hypercall completed successfully:

- The device, referred to by the SID/BDF in dev, has been assigned to the Protection Domain (PD) referred to by { $PD_{CURRENT}$, SEL_{OBJ} pd }, such that DMA transactions of that device will be translated by the DMA page table tbl of that PD.
- DMA transactions of that device will be managed using the SMMU resources encoded in dev. Prior users of those SMMU resources have been unconfigured.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_HYP

- The hypercall was not issued from the Root Protection Domain (PD_{ROOT}).

BAD_DEV

- { PD_{NOVA}, SEL_{MEM} smmu } did not refer to the physical address of an SMMU device.

BAD_CAP

- { $PD_{CURRENT}$, SEL_{OBJ} pd } did not refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) or that capability had insufficient permissions.

BAD_PAR

- At least one of the parameters dev or tbl was not valid.

*See the architecture-specific binding for encoding details.

5 Booting

5.1 Microhypervisor

5.1.1 ELF Image Loading

The bootloader must load the NOVA microhypervisor into physical memory according to the physical addresses (`PhysAddr`) and memory sizes (`MemSiz`) of all loadable (`PT_LOAD`) program segments defined in the NOVA microhypervisor ELF image. The following is an example:

```
readelf -l hypervisor.elf

Elf file type is EXEC (Executable file)
Entry point 0x48000000
There are 2 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz             Flags  Align
  LOAD           0x00000000000000b0 0x0000000048000000 0x0000000048000000
                 0x0000000000000268 0x0000000000001000 RWE    0x8
  LOAD           0x0000000000000800 0x0000ff8000001000 0x0000000048001000
                 0x000000000000e960 0x0000000000fff000 RWE    0x800
```

If the physical address range defined in the ELF image is suboptimal for a particular platform, the bootloader may optionally shift all loadable program segments lower or higher in physical memory, by applying an offset, subject to the following constraints:

- The same offset must be applied to each loadable program segment and to the entry point.
- The offset must be a multiple of 2MiB, i.e. $\mathtt{PhysAddr}_{\mathrm{NEW}} = \mathtt{PhysAddr}_{\mathrm{ELF}} \pm \mathtt{n} \times \mathtt{2MiB}$.
- The entire physical memory region occupied by the NOVA microhypervisor must be RAM.

After loading the NOVA microhypervisor into physical memory, the bootloader must invoke the entry point of the ELF image with architecture-specific preconditions (ARM, x86).

5.1.2 Platform Resource Access

Possession of a PD Object Capability ($\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{PD}}}$) for $\mathtt{PD}_{\mathtt{NOVA}}$ allows the caller to invoke the `ctrl_pd` hypercall to take resources from the NOVA Protection Domain and grant them to another Protection Domain.

The following capabilities can be taken from the NOVA Protection Domain ($\mathtt{PD}_{\mathtt{NOVA}}$):

Physical Memory

{ $\mathtt{PD}_{\mathtt{NOVA}}$, $\mathtt{SEL}_{\mathtt{MEM}}$ $\mathtt{0...PHYS}_{\mathtt{NUM}}\mathtt{-1}$ } refer to $\mathtt{CAP}_{\mathtt{MEM}}$ for physical memory pages, where $\mathtt{PHYS}_{\mathtt{NUM}}$ is the number of physical memory pages supported by the platform. Physical memory regions protected by the NOVA microhypervisor (ARM, x86) cannot be taken.

Interrupt Semaphores

{ $\mathtt{PD}_{\mathtt{NOVA}}$, $\mathtt{SEL}_{\mathtt{OBJ}}$ $\mathtt{1024...1024+INT}_{\mathtt{NUM}}\mathtt{-1}$ } refer to $\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{SM}}}$ for interrupt semaphores, where $\mathtt{INT}_{\mathtt{NUM}}$ is the number of supported interrupts, as conveyed by the HIP. These capabilities can be used with the `assign_int` and `ctrl_sm` hypercalls.

Console Signaling Semaphore

{ $\mathtt{PD}_{\mathtt{NOVA}}$, $\mathtt{SEL}_{\mathtt{OBJ}}$ $\mathtt{SEL}_{\mathtt{NUM}}\mathtt{-1}$ } refers to a $\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{SM}}}$ for the signaling semaphore of the NOVA memory-buffer console. This capability can be used with the `ctrl_sm` hypercall.

5.2 Root Protection Domain

After the NOVA microhypervisor has initialized the system, it creates the following initial kernel objects:

- PD_{ROOT} – the Root Protection Domain
- EC_{ROOT} – the Root Execution Context (executing in PD_{ROOT})
- SC_{ROOT} – the Root Scheduling Context (bound to EC_{ROOT})

The Root Protection Domain (PD_{ROOT}) is responsible for bootstrapping the other components of the user-mode framework by creating additional kernel objects, loading additional images, assigning resources, etc.

5.2.1 Initial Configuration

Prior to invoking the entry point of the Root Protection Domain (PD_{ROOT}) ELF image, using the Root Execution Context (EC_{ROOT}), the NOVA microhypervisor sets up PD_{ROOT} as follows.

5.2.1.1 Object Space

The object space contains the following initial capabilities:

- { PD_{ROOT}, SEL_{OBJ} SEL_{NUM}-1 } refers to a PD Object Capability ($CAP_{OBJ_{PD}}$) for PD_{NOVA}.
- { PD_{ROOT}, SEL_{OBJ} SEL_{NUM}-2 } refers to a PD Object Capability ($CAP_{OBJ_{PD}}$) for PD_{ROOT}.
- { PD_{ROOT}, SEL_{OBJ} SEL_{NUM}-3 } refers to a EC Object Capability ($CAP_{OBJ_{EC}}$) for EC_{ROOT}.
- { PD_{ROOT}, SEL_{OBJ} SEL_{NUM}-4 } refers to a SC Object Capability ($CAP_{OBJ_{SC}}$) for SC_{ROOT}.

All other { PD_{ROOT}, SEL_{OBJ} } refer to a Null Capability ($CAP_{\emptyset}$).

The value of SEL_{NUM} is conveyed in the Hypervisor Information Page (HIP).

5.2.1.2 Memory Space

ELF Program Segments

The microhypervisor maps the root protection domain into virtual memory according to the virtual addresses (`VirtAddr`) and memory sizes (`MemSiz`) of all loadable (`PT_LOAD`) program segments defined in the root protection domain ELF image.

Hypervisor Information Page

The microhypervisor maps the Hypervisor Information Page (HIP) into the memory space 4KB below the end of user-accessible virtual memory. The virtual address of the HIP is passed to EC_{ROOT} during startup.

UTCB

The microhypervisor maps the User Thread Control Block of EC_{ROOT} into the memory space 4KB below the address of the HIP.

All other { PD_{ROOT}, SEL_{MEM} } refer to a Null Capability ($CAP_{\emptyset}$).

5.3 Hypervisor Information Page

The Hypervisor Information Page (HIP) conveys information about the platform and configuration to the Root Protection Domain (PD_{ROOT}) and has the following layout:

63 … 48	47 … 32	31 … 16	15 … 0	
Architecture-Dependent				+Length
Architecture-Dependent				+0x70
Features		INT_{NUM}	CPU_{NUM}	+0x68
$SEL_{GST/NOVA}$	$SEL_{GST/ARCH}$	$SEL_{HST/NOVA}$	$SEL_{HST/ARCH}$	+0x60
SEL_{NUM}				+0x58
Timer Frequency				+0x50
UEFI Desc Version	UEFI Desc Size	UEFI Memory Map Size		+0x48
UEFI Memory Map Address				+0x40
ACPI RSDP Address				+0x38
ROOT End Address				+0x30
ROOT Start Address				+0x28
MBUF End Address				+0x20
MBUF Start Address				+0x18
NOVA End Address				+0x10
NOVA Start Address				+0x08
Length	Checksum	Signature		+0x00

All HIP fields are unsigned values, unless stated otherwise, and have the following meaning:

Signature

The value `0x41564f4e` identifies the NOVA microhypervisor.

Checksum

The checksum is valid if 16bit-wise addition of the entire HIP contents produces a value of `0`.

Length

Length of the entire HIP in bytes.

NOVA Start/End Address

Physical start and end address of the NOVA microhypervisor image.

MBUF Start/End Address

Physical start and end address of the memory buffer console region (see C.1).

ROOT Start/End Address

Physical start and end address of the root protection domain image.

ACPI RSDP Address

Physical address of the ACPI Root System Description Pointer (`0xffffffffffffffff` if not present).

UEFI Memory Map Address

Physical address of the UEFI Memory Map (`0xffffffffffffffff` if not present).

UEFI Memory Map Size

Total size of the `UEFI` Memory Map (`0` if not present).

UEFI Desc Size

`UEFI` Memory Descriptor Size (`0` if not present).

UEFI Desc Version

`UEFI` Memory Descriptor Version (`0` if not present).

Timer Frequency

Timer tick frequency in Hz.

SEL_{NUM}

Total number of capability selectors in each object space.

$\text{SEL}_{\text{HST/ARCH}}$

Number of capability selectors required for handling architectual host events. (ARM, x86)

$\text{SEL}_{\text{HST/NOVA}}$

Number of additional capability selectors required for handling microhypervisor host events. (ARM, x86)

$\text{SEL}_{\text{GST/ARCH}}$

Number of capability selectors required for handling architectual guest events. (ARM, x86)

$\text{SEL}_{\text{GST/NOVA}}$

Number of additional capability selectors required for handling microhypervisor guest events. (ARM, x86)

CPU_{NUM}

Total number of CPUs that are online.

INT_{NUM}

Total number of interrupts that can be used via interrupt semaphores.

Features

Supported platform features.

Architecture-Dependent

Architecture-dependent part. (ARM, x86)

Part IV

Application Binary Interface

6 ABI aarch64

6.1 Boot State

6.1.1 NOVA Microhypervisor

The bootloader must set up the CPU register state according to one of the launch types listed below when it transfers control to the NOVA microhypervisor. Furthermore, the following preconditions must be satisfied:

- The CPU must execute in EL2 (hypervisor mode) or in EL3 (monitor mode).
- Paging (MMU) must be disabled (`SCTLR_ELx.M=0`) or must use an identity (1:1) mapping.
- Interrupts must be disabled (`PSTATE.DAIF=0b1111`).
- The address range corresponding to the microhypervisor image must be clean to the Point of Coherence.
- All DMA activity targeting the physical memory region occupied by the microhypervisor must be quiesced. That physical memory region should also be protected against DMA accesses on systems with an SMMU.

6.1.1.1 Multiboot v2 Launch

Only this launch type supports 64-bit UEFI platforms.

Register	Value / Description
`IP`	Physical address of the NOVA Protection Domain (PD_{NOVA}) ELF image entry point
`X0`	Multiboot v2 magic value (`0x36d76289`) [6]
`X1`	Physical address of the Multiboot v2 information structure [6]
`Other`	~

The NOVA microhypervisor consumes the following multiboot tags, if present: `1`, `3`, `12`, `20`.

6.1.1.2 Multiboot v1 Launch

Register	Value / Description
`IP`	Physical address of the NOVA Protection Domain (PD_{NOVA}) ELF image entry point
`X0`	Multiboot v1 magic value (`0x2badb002`) [5]
`X1`	Physical address of the Multiboot v1 information structure [5]
`Other`	~

The NOVA microhypervisor consumes the following multiboot flags, if present: `2`, `3`.

6.1.1.3 Legacy Launch

Register	Value / Description
`IP`	Physical address of the NOVA Protection Domain (PD_{NOVA}) ELF image entry point
`X0`	Physical address of the Flattened Device Tree [4] (FDT) for the system[†]
`X1`	Physical address of the Root Protection Domain (PD_{ROOT}) ELF image
`Other`	~

[†]Due to its alignment constraint, a valid FDT address will never be equal to a Multiboot magic value.

6.1.2 Root Protection Domain

The NOVA microhypervisor sets up the CPU register state as follows when it transfers control to the Root Execution Context (EC_{ROOT}):

Register	Value / Description
IP	Virtual address of the Root Protection Domain (PD_{ROOT}) ELF image entry point
SP	Virtual address of the Hypervisor Information Page (HIP)
X0	X0 at boot time †
X1	X1 at boot time †
X2	X2 at boot time †
Other	~

† The register contains the preserved original value from the point when control was transferred from the bootloader to the microhypervisor.

6.2 Physical Memory

6.2.1 Memory Map

The Root Protection Domain ($\mathrm{PD}_{\mathrm{ROOT}}$) can obtain a list of available/reserved memory regions as follows:

- On platforms using Unified Extensible Firmware Interface [11], by parsing the UEFI memory map.
- On platforms using Flattened Device Tree [4], by parsing the FDT.

6.2.2 Protected Regions

The following regions of physical memory are protected by the NOVA microhypervisor and are therefore inaccessible to user-mode applications:

- Physical memory occupied by the NOVA microhypervisor (conveyed via `HIP`).
- Physical memory occupied by GICD, GICR, GICC, GICH devices (conveyed via `FDT`).
- Physical memory occupied by SMMU devices (conveyed via `FDT`).

6.3 Virtual Memory

The accessible virtual memory range for user-mode applications is `0` – `0x7fffffffff`.

6.3.1 Cacheability Attributes

Number	$\mathtt{ATTR}_{\mathtt{CA}}$	Description
`0x0`	`DEV`	Device
`0x1`	`DEV_E`	Device, Early Ack
`0x2`	`DEV_RE`	Device, Early Ack, Reordering
`0x3`	`DEV_GRE`	Device, Early Ack, Reordering, Gathering
`0x4`	-	*reserved*
`0x5`	`MEM_NC`	Memory, Inner/Outer Non-Cacheable
`0x6`	`MEM_WT`	Memory, Inner/Outer Write-Through
`0x7`	`MEM_WB`	Memory, Inner/Outer Write-Back

Please refer to [2] for details on the architectural behavior.

6.3.2 Shareability Attributes

Number	$\mathtt{ATTR}_{\mathtt{SH}}$	Description
`0x0`	`NONE`	Not Shareable
`0x1`	-	*reserved*
`0x2`	`OUTER`	Outer Shareable
`0x3`	`INNER`	Inner Shareable

Please refer to [2] for details on the architectural behavior.

6.4 Event-Specific Capability Selectors

For the delivery of exception/intercept messages, the microhypervisor performs an implicit portal traversal.

The selector for the destination portal (SEL_{OBJ}) is determined by adding the exception/intercept number to SEL_{EVT} of the affected execution context and that selector must refer to a PT Object Capability ($CAP_{OBJ_{PT}}$).

6.4.1 Architectural Events

SEL_{OBJ}	Exception / Intercept	SEL_{OBJ}	Exception / Intercept
SEL_{EVT} + 0x0	Unknown Reason	SEL_{EVT} + 0x20	Instruction Abort (lower EL)
SEL_{EVT} + 0x1	Trapped WFI or WFE	SEL_{EVT} + 0x21	Instruction Abort (same EL)
SEL_{EVT} + 0x2	reserved	SEL_{EVT} + 0x22	PC Alignment Fault
SEL_{EVT} + 0x3	Trapped MCR or MRC	SEL_{EVT} + 0x23	reserved
SEL_{EVT} + 0x4	Trapped MCRR or MRRC	SEL_{EVT} + 0x24	Data Abort (lower EL)
SEL_{EVT} + 0x5	Trapped MCR or MRC	SEL_{EVT} + 0x25	Data Abort (same EL)
SEL_{EVT} + 0x6	Trapped LDC or STC	SEL_{EVT} + 0x26	SP Alignment Fault
SEL_{EVT} + 0x7	SVE, SIMD, FPU	SEL_{EVT} + 0x27	reserved
SEL_{EVT} + 0x8	Trapped VMRS Access	SEL_{EVT} + 0x28	Trapped FPU (AArch32)
SEL_{EVT} + 0x9	Trapped PAuth Instruction	SEL_{EVT} + 0x29	reserved
SEL_{EVT} + 0xa	reserved	SEL_{EVT} + 0x2a	reserved
SEL_{EVT} + 0xb	reserved	SEL_{EVT} + 0x2b	reserved
SEL_{EVT} + 0xc	Trapped MRRC	SEL_{EVT} + 0x2c	Trapped FPU (AArch64)
SEL_{EVT} + 0xd	reserved	SEL_{EVT} + 0x2d	reserved
SEL_{EVT} + 0xe	Illegal Execution State	SEL_{EVT} + 0x2e	reserved
SEL_{EVT} + 0xf	reserved	SEL_{EVT} + 0x2f	SError
SEL_{EVT} + 0x10	reserved	SEL_{EVT} + 0x30	Breakpoint (lower EL)
SEL_{EVT} + 0x11	SVC (from AArch32 State)*	SEL_{EVT} + 0x31	Breakpoint (same EL)
SEL_{EVT} + 0x12	HVC (from AArch32 State)	SEL_{EVT} + 0x32	Software Step (lower EL)
SEL_{EVT} + 0x13	SMC (from AArch32 State)	SEL_{EVT} + 0x33	Software Step (same EL)
SEL_{EVT} + 0x14	reserved	SEL_{EVT} + 0x34	Watchpoint (lower EL)
SEL_{EVT} + 0x15	SVC (from AArch64 State)*	SEL_{EVT} + 0x35	Watchpoint (same EL)
SEL_{EVT} + 0x16	HVC (from AArch64 State)	SEL_{EVT} + 0x36	reserved
SEL_{EVT} + 0x17	SMC (from AArch64 State)	SEL_{EVT} + 0x37	reserved
SEL_{EVT} + 0x18	Trapped MSR or MRS	SEL_{EVT} + 0x38	BKPT (AArch32)
SEL_{EVT} + 0x19	Trapped SVE	SEL_{EVT} + 0x39	reserved
SEL_{EVT} + 0x1a	Trapped ERET	SEL_{EVT} + 0x3a	Vector Catch (AArch32)
SEL_{EVT} + 0x1b	reserved	SEL_{EVT} + 0x3b	reserved
SEL_{EVT} + 0x1c	reserved	SEL_{EVT} + 0x3c	BRK (AArch64)
SEL_{EVT} + 0x1d	reserved	SEL_{EVT} + 0x3d	reserved
SEL_{EVT} + 0x1e	reserved	SEL_{EVT} + 0x3e	reserved
SEL_{EVT} + 0x1f	reserved	SEL_{EVT} + 0x3f	reserved

Please refer to [2] for more details on each of these events.

6.4.2 Microhypervisor Events

SEL_{OBJ}	Event
SEL_{EVT} + SEL_{ARCH} + 0x0	Startup
SEL_{EVT} + SEL_{ARCH} + 0x1	Recall
SEL_{EVT} + SEL_{ARCH} + 0x2	Virtual Timer

The value of SEL_{ARCH} depends on the origin of the event:

- SEL_{ARCH} = $SEL_{HST/ARCH}$ (0x40) for events that occurred in the host.
- SEL_{ARCH} = $SEL_{GST/ARCH}$ (0x40) for events that occurred in the guest.

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

6.5 Architecture-Dependent Structures

6.5.1 Hypervisor Information Page

63–32	31–16	15–0	+Length
~	CTX_{NUM}	SMG_{NUM}	Arch+0x00

SMG_{NUM}

Number of SMMU stream mapping groups.

CTX_{NUM}

Number of SMMU translation contexts.

6.5.2 User Thread Control Block

-		VMCR	ELRSR	+0x2d0	GIC
AP1R3	AP1R2	AP1R1	AP1R0	+0x2c0	
AP0R3	AP0R2	AP0R1	AP0R0	+0x2b0	
LR15		LR14		+0x2a0	
LR13		LR12		+0x290	
LR11		LR10		+0x280	
LR9		LR8		+0x270	
LR7		LR6		+0x260	
LR5		LR4		+0x250	
LR3		LR2		+0x240	
LR1		LR0		+0x230	
CNTVOFF_EL2		CNTKCTL_EL1		+0x220	TMR
CNTV_CTL_EL0		CNTV_CVAL_EL0		+0x210	
-		HPFAR_EL2		+0x200	EL2
FAR_EL2		ESR_EL2		+0x1f0	
SPSR_EL2		ELR_EL2		+0x1e0	
VMPIDR_EL2		VPIDR_EL2		+0x1d0	
HCRX_EL2		HCR_EL2		+0x1c0	
-		MDSCR_EL1		+0x1b0	EL1
SCTLR_EL1		VBAR_EL1		+0x1a0	
AMAIR_EL1		MAIR_EL1		+0x190	
TCR_EL1		TTBR1_EL1		+0x180	
TTBR0_EL1		AFSR1_EL1		+0x170	
AFSR0_EL1		FAR_EL1		+0x160	
ESR_EL1		SPSR_EL1		+0x150	
ELR_EL1		CONTEXTIDR_EL1		+0x140	
TPIDR_EL1		SP_EL1		+0x130	
-		IFSR	DACR	+0x120	A32
SPSR_und	SPSR_irq	SPSR_fiq	SPSR_abt	+0x110	
TPIDRRO_EL0		TPIDR_EL0		+0x100	EL0
SP_EL0		X30		+0x0f0	
X29		X28		+0x0e0	
X27		X26		+0x0d0	
X25		X24		+0x0c0	
X23		X22		+0x0b0	
X21		X20		+0x0a0	
X19		X18		+0x090	
X17		X16		+0x080	
X15		X14		+0x070	
X13		X12		+0x060	
X11		X10		+0x050	
X9		X8		+0x040	
X7		X6		+0x030	
X5		X4		+0x020	
X3		X2		+0x010	
X1		X0		+0x000	
48 32 16 0		48 32 16 0			

6.5.3 Message Transfer Descriptor

The Message Transfer Descriptor (MTD), which controls the subset of the architectural state transferred during exceptions and intercepts, as described in Section 3.4.2, has the following layout:

GIC	TMR	-	EL2_HPFAR	EL2_ESR_FAR	EL2_ELR_SPSR	EL2_IDR	EL2_HCR	-	EL1_MDSCR	EL1_SCTLR	EL1_VBAR	EL1_MAIR	EL1_TCR	EL1_TTBR	EL1_AFSR	EL1_ESR_FAR	EL1_ELR_SPSR	EL1_IDR	EL1_SP	-	A32_DACR_IFSR	A32_SPSR	-	EL0_IDR	EL0_SP	FPR	GPR	ICI	POISON
31	30		27	26	25	24	23		20	19	18	17	16	15	14	13	12	11	10		8	7		5	4	3	2	1	0

Each MTD bit controls the transfer of the listed architectural state to/from the respective fields in the UTCB (6.5.2) as follows:

- State with access `r` can be read from the architectural state into the UTCB.
- State with access `w` can be written from the UTCB into the architectural state.

MTD Bit	Access	Host Exception State	Guest Intercept State
POISON	w	Kills the EC	Kills the EC
ICI[†]	w	Invalidates the entire I-Cache	Invalidates the entire I-Cache
GPR	rw	X0 ... X30	X0 ... X30
EL0_SP	rw	SP_EL0	SP_EL0
EL0_IDR	rw	TPIDR_EL0, TPIDRRO_EL0	TPIDR_EL0, TPIDRRO_EL0
A32_SPSR	rw	-	SPSR_ABT, SPSR_FIQ, SPSR_IRQ, SPSR_UND
A32_DACR_IFSR	rw	-	DACR, IFSR
EL1_SP	rw	-	SP_EL1
EL1_IDR	rw	-	TPIDR_EL1, CONTEXTIDR_EL1
EL1_ELR_SPSR	rw	-	ELR_EL1, SPSR_EL1
EL1_ESR_FAR	rw	-	ESR_EL1, FAR_EL1
EL1_AFSR	rw	-	AFSR0_EL1, AFSR1_EL1
EL1_TTBR	rw	-	TTBR0_EL1, TTBR1_EL1
EL1_TCR	rw	-	TCR_EL1
EL1_MAIR	rw	-	MAIR_EL1, AMAIR_EL1
EL1_VBAR	rw	-	VBAR_EL1
EL1_SCTLR	rw	-	SCTLR_EL1
EL1_MDSCR	rw	-	MDSCR_EL1
EL2_HCR	rw	-	HCR_EL2, HCRX_EL2
EL2_IDR	rw	-	VPIDR_EL2, VMPIDR_EL2
EL2_ELR_SPSR	rw	ELR_EL2, SPSR_EL2	ELR_EL2, SPSR_EL2
EL2_ESR_FAR	r	ESR_EL2, FAR_EL2	ESR_EL2, FAR_EL2
EL2_HPFAR	r	-	HPFAR_EL2
TMR	rw	-	CNTV_CVAL_EL0, CNTV_CTL_EL0 CNTKCTL_EL1, CNTVOFF_EL2
GIC	rw r	-	LR0 ... LR15, APxR0 ... APxR3 ELRSR, VMCR

[†]Only affects a VIPT instruction cache of the local core. Has no effect on PIPT instruction caches, data caches, or caches of other cores.

6.6 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 `svc #0` instruction [2].

The hypercall identifier consists of the hypercall number and hypercall-specific flags, as illustrated in Figure 6.1.

flags	number
7 … 4	3 … 0

Figure 6.1: Hypercall Identifier

The status code returned from a hypercall has the format shown in Figure 6.2.

status
7 … 0

Figure 6.2: 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.

IPC Call

		ipc_call $\longrightarrow$		
$pt_{[63-8]}$ $hypercall_{[7-0]}$	`X0`		`X0`	$status_{[7-0]}$
$mtd_{[31-0]}$	`X1`		`X1`	$mtd_{[31-0]}$
–	`IP`	`svc #0`	`IP`	`IP+4`

IPC Reply

		ipc_reply $\longrightarrow$		
$hypercall_{[7-0]}$	`X0`		`X0`	pid
$mtd_{[31-0]}$	`X1`		`X1`	$mtd_{[31-0]}$
–	`IP`	`svc #0`	`IP`	Portal `IP`

Create Protection Domain

		create_pd $\longrightarrow$		
$sel_{[63-8]}$ $hypercall_{[7-0]}$	`X0`		`X0`	$status_{[7-0]}$
own	`X1`		`X1`	≡
–	`IP`	`svc #0`	`IP`	`IP+4`

Create Execution Context

		create_ec $\longrightarrow$		
$sel_{[63-8]}$ $hypercall_{[7-0]}$	`X0`		`X0`	$status_{[7-0]}$
own	`X1`		`X1`	≡
$utcb_{[63-12]}$ $cpu_{[11-0]}$	`X2`		`X2`	≡
sp	`X3`		`X3`	≡
evt	`X4`		`X4`	≡
–	`IP`	`svc #0`	`IP`	`IP+4`

Create Scheduling Context

sel$_{[63-8]}$ hypercall$_{[7-0]}$	X0	create_sc $\longrightarrow$	X0	status$_{[7-0]}$
own	X1		X1	≡
ec	X2		X2	≡
quantum$_{[31-12]}$ prio$_{[6-0]}$	X3		X3	≡
–	IP	svc #0	IP	IP+4

Create Portal

sel$_{[63-8]}$ hypercall$_{[7-0]}$	X0	create_pt $\longrightarrow$	X0	status$_{[7-0]}$
own	X1		X1	≡
ec	X2		X2	≡
ip	X3		X3	≡
–	IP	svc #0	IP	IP+4

Create Semaphore

sel$_{[63-8]}$ hypercall$_{[7-0]}$	X0	create_sm $\longrightarrow$	X0	status$_{[7-0]}$
own	X1		X1	≡
cnt	X2		X2	≡
–	IP	svc #0	IP	IP+4

Control Protection Domain

spd$_{[63-8]}$ hypercall$_{[7-0]}$	X0	ctrl_pd $\longrightarrow$	X0	status$_{[7-0]}$
dpd	X1		X1	≡
src$_{[63-12]}$ ord$_{[6-2]}$ spc$_{[1-0]}$	X2		X2	≡
dst$_{[63-12]}$ sh$_{[11-10]}$ ca$_{[9-7]}$ pmm$_{[6-2]}$ tbl$_{[1-0]}$	X3		X3	≡
–	IP	svc #0	IP	IP+4

Control Execution Context

ec$_{[63-8]}$ hypercall$_{[7-0]}$	X0	ctrl_ec $\longrightarrow$	X0	status$_{[7-0]}$
–	IP	svc #0	IP	IP+4

Control Scheduling Context

sc$_{[63-8]}$ hypercall$_{[7-0]}$	X0	ctrl_sc $\longrightarrow$	X0	status$_{[7-0]}$
–	X1		X1	ticks
–	IP	svc #0	IP	IP+4

Control Portal

pt$_{[63-8]}$ hypercall$_{[7-0]}$	X0	ctrl_pt $\longrightarrow$	X0	status$_{[7-0]}$
pid	X1		X1	≡
mtd$_{[31-0]}$	X2		X2	≡
–	IP	svc #0	IP	IP+4

Control Semaphore

$sm_{[63-8]}$ $hypercall_{[7-0]}$	X0	**ctrl_sm** $\longrightarrow$	X0	$status_{[7-0]}$
ticks	X1		X1	≡
–	IP	svc #0	IP	IP+4

Control Hardware

$hypercall_{[7-0]}$	X0	**ctrl_hw** $\longrightarrow$	X0	$status_{[7-0]}$
p0	X1		X1	p0
p1	X2		X2	p1
p2	X3		X3	p2
p3	X4		X4	p3
p4	X5		X5	≡
p5	X6		X6	≡
p6	X7		X7	≡
–	IP	svc #0	IP	IP+4

Assign Interrupt

$sm_{[63-8]}$ $hypercall_{[7-0]}$	X0	**assign_int** $\longrightarrow$	X0	$status_{[7-0]}$
cpu	X1		X1	$msi\_addr_{[31-0]}$
$sid_{[15-0]}$	X2		X2	$msi\_data_{[15-0]}$
–	IP	svc #0	IP	IP+4

Assign Device

$pd_{[63-8]}$ $hypercall_{[7-0]}$	X0	**assign_dev** $\longrightarrow$	X0	$status_{[7-0]}$
$smmu_{[63-12]}$ $tbl_{[1-0]}$	X1		X1	≡
$ctx_{[31-24]}$ $smg_{[23-16]}$ $sid_{[15-0]}$	X2		X2	≡
–	IP	svc #0	IP	IP+4

7 ABI x86-64

7.1 Boot State

7.1.1 NOVA Microhypervisor

The bootloader must set up the CPU register state according to one of the launch types listed below when it transfers control to the NOVA microhypervisor. Furthermore, the following preconditions must be satisfied:

- The CPU state must conform to a machine state defined in the Multiboot Specification v2 [6] or v1 [5].
- All DMA activity targeting the physical memory region occupied by the microhypervisor must be quiesced. That physical memory region should also be protected against DMA accesses on systems with an IOMMU.

7.1.1.1 Multiboot v2 Launch

Only this launch type supports 64-bit UEFI platforms.

Register	Value / Description
EIP	Physical address of the NOVA Protection Domain ($\mathrm{PD}_{\mathtt{NOVA}}$) ELF image entry point
EAX	Multiboot v2 magic value (`0x36d76289`) [6]
EBX	Physical address of the Multiboot v2 information structure [6]
Other	~

The NOVA microhypervisor consumes the following multiboot tags, if present: `1`, `3`, `12`, `20`.

7.1.1.2 Multiboot v1 Launch

Register	Value / Description
EIP	Physical address of the NOVA Protection Domain ($\mathrm{PD}_{\mathtt{NOVA}}$) ELF image entry point
EAX	Multiboot v1 magic value (`0x2badb002`) [5]
EBX	Physical address of the Multiboot v1 information structure [5]
Other	~

The NOVA microhypervisor consumes the following multiboot flags, if present: `2`, `3`.

7.1.2 Root Protection Domain

The NOVA microhypervisor sets up the CPU register state as follows when it transfers control to the Root Execution Context ($\mathrm{EC}_{\mathtt{ROOT}}$):

Register	Value / Description
RIP	Virtual address of the Root Protection Domain ($\mathrm{PD}_{\mathtt{ROOT}}$) ELF image entry point
RSP	Virtual address of the Hypervisor Information Page (HIP)
RDI	EAX at boot time †
RSI	EBX at boot time †
Other	~

† The register contains the preserved original value from the point when control was transferred from the bootloader to the microhypervisor.

7.2 Physical Memory

7.2.1 Memory Map

The Root Protection Domain ($\mathrm{PD}_{\mathrm{ROOT}}$) can obtain a list of available/reserved memory regions as follows:

- On platforms using Multiboot v2 (UEFI boot services enabled), by parsing the UEFI memory map [11].
- On platforms using Multiboot v2, by parsing the Multiboot v2 memory map [6].
- On platforms using Multiboot v1, by parsing the Multiboot v1 memory map [5].

7.2.2 Protected Regions

The following regions of physical memory are protected by the NOVA microhypervisor and are therefore inaccessible to user-mode applications:

- Physical memory occupied by the NOVA microhypervisor (conveyed via `HIP`).
- Physical memory occupied by Local APIC and I/O APIC devices (conveyed via `ACPI` `MADT`).
- Physical memory occupied by IOMMU devices (conveyed via `ACPI` `DMAR`).
- Physical memory occupied by firmware runtime services (conveyed via `UEFI` memory map).

7.3 Virtual Memory

The accessible virtual memory range for user-mode applications is `0` – `0x7fffffffffff`.

7.3.1 Cacheability Attributes

Number	$\mathtt{ATTR}_{\mathtt{CA}}$	Description
`0x0`	`WB`	Write Back
`0x1`	`WT`	Write Through
`0x2`	`WC`	Write Combining
`0x3`	`UC`	Strong Uncacheable
`0x4`	`WP`	Write Protected

Please refer to [1, 3] for details on the architectural behavior.

7.3.2 Shareability Attributes

Number	$\mathtt{ATTR}_{\mathtt{SH}}$	Description
`0x0`	`UNUSED`	Always use this value

7.4 Event-Specific Capability Selectors

For the delivery of exception/intercept messages, the microhypervisor performs an implicit portal traversal.

The selector for the destination portal (SEL_{OBJ}) is determined by adding the exception/intercept number to SEL_{EVT} of the affected execution context and that selector must refer to a PT Object Capability ($CAP_{OBJ_{PT}}$).

7.4.1 Architectural Events

Host Exceptions

SEL_{OBJ}	Exception	SEL_{OBJ}	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*
SEL_{EVT} + 0x3	#BP	SEL_{EVT} + 0x13	#XM
SEL_{EVT} + 0x4	#OF	SEL_{EVT} + 0x14	#VE
SEL_{EVT} + 0x5	#BR	SEL_{EVT} + 0x15	#CP
SEL_{EVT} + 0x6	#UD	SEL_{EVT} + 0x16	reserved
SEL_{EVT} + 0x7	#NM*	SEL_{EVT} + 0x17	reserved
SEL_{EVT} + 0x8	#DF*	SEL_{EVT} + 0x18	reserved
SEL_{EVT} + 0x9	reserved	SEL_{EVT} + 0x19	reserved
SEL_{EVT} + 0xa	#TS*	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	reserved
SEL_{EVT} + 0xf	reserved	SEL_{EVT} + 0x1f	reserved

*These events may be handled by the microhypervisor, in which case they will not cause portal traversals.
†These events may be force-enabled by the microhypervisor, in which case they will cause portal traversals.

Guest Intercepts (VMX)

SEL_{OBJ}	Intercept
SEL_{EVT} + 0x0	Exception or NMI*
SEL_{EVT} + 0x1	External Interrupt*
SEL_{EVT} + 0x2	Triple Fault†
SEL_{EVT} + 0x3	INIT†
SEL_{EVT} + 0x4	SIPI†
SEL_{EVT} + 0x5	I/O SMI
SEL_{EVT} + 0x6	Other SMI
SEL_{EVT} + 0x7	Interrupt Window
SEL_{EVT} + 0x8	NMI Window
SEL_{EVT} + 0x9	Task Switch†
SEL_{EVT} + 0xa	CPUID†
SEL_{EVT} + 0xb	GETSEC†
SEL_{EVT} + 0xc	HLT†
SEL_{EVT} + 0xd	INVD†
SEL_{EVT} + 0xe	INVLPG*
SEL_{EVT} + 0xf	RDPMC
SEL_{EVT} + 0x10	RDTSC
SEL_{EVT} + 0x11	RSM
SEL_{EVT} + 0x12	VMCALL
SEL_{EVT} + 0x13	VMCLEAR
SEL_{EVT} + 0x14	VMLAUNCH
SEL_{EVT} + 0x15	VMPTRLD
SEL_{EVT} + 0x16	VMPTRST
SEL_{EVT} + 0x17	VMREAD
SEL_{EVT} + 0x18	VMRESUME
SEL_{EVT} + 0x19	VMWRITE
SEL_{EVT} + 0x1a	VMXOFF
SEL_{EVT} + 0x1b	VMXON
SEL_{EVT} + 0x1c	CR Access*
SEL_{EVT} + 0x1d	DR Access
SEL_{EVT} + 0x1e	I/O Access†
SEL_{EVT} + 0x1f	RDMSR†
SEL_{EVT} + 0x20	WRMSR†
SEL_{EVT} + 0x21	VM Entry Failure (State)†
SEL_{EVT} + 0x22	VM Entry Failure (MSR)
SEL_{EVT} + 0x23	reserved
SEL_{EVT} + 0x24	MWAIT
SEL_{EVT} + 0x25	MTF
SEL_{EVT} + 0x26	reserved
SEL_{EVT} + 0x27	MONITOR
SEL_{EVT} + 0x28	PAUSE
SEL_{EVT} + 0x29	VM Entry Failure (MCE)
SEL_{EVT} + 0x2a	reserved
SEL_{EVT} + 0x2b	TPR Below Threshold
SEL_{EVT} + 0x2c	APIC Access
SEL_{EVT} + 0x2d	Virtualized EOI
SEL_{EVT} + 0x2e	GDTR/IDTR Access
SEL_{EVT} + 0x2f	LDTR/TR Access
SEL_{EVT} + 0x30	EPT Violation†
SEL_{EVT} + 0x31	EPT Misconfiguration
SEL_{EVT} + 0x32	INVEPT
SEL_{EVT} + 0x33	RDTSCP
SEL_{EVT} + 0x34	Preemption Timer
SEL_{EVT} + 0x35	INVVPID
SEL_{EVT} + 0x36	WBINVD, WBNOINVD
SEL_{EVT} + 0x37	XSETBV
SEL_{EVT} + 0x38	APIC Write
SEL_{EVT} + 0x39	RDRAND
SEL_{EVT} + 0x3a	INVPCID
SEL_{EVT} + 0x3b	VMFUNC
SEL_{EVT} + 0x3c	ENCLS
SEL_{EVT} + 0x3d	RDSEED
SEL_{EVT} + 0x3e	PML Log Full
SEL_{EVT} + 0x3f	XSAVES
SEL_{EVT} + 0x40	XRSTORS
SEL_{EVT} + 0x41	reserved
SEL_{EVT} + 0x42	SPP Miss / Misconfiguration
SEL_{EVT} + 0x43	UMWAIT
SEL_{EVT} + 0x44	TPAUSE
SEL_{EVT} + 0x45	LOADIWKEY
SEL_{EVT} + 0x46	reserved
SEL_{EVT} + 0x47	reserved
SEL_{EVT} + 0x48	ENQCMD PASID Failure
SEL_{EVT} + 0x49	ENQCMDS PASID Failure
SEL_{EVT} + 0x4a	Bus Lock
SEL_{EVT} + 0x4b	Notify Window
SEL_{EVT} + 0x4c	SEAMCALL
SEL_{EVT} + 0x4d	TDCALL
SEL_{EVT} + 0x4e	reserved
SEL_{EVT} + 0x4f	reserved

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

7.4.2 Microhypervisor Events

SEL_{OBJ}	Event
SEL_{EVT} + SEL_{ARCH} + 0x0	Startup
SEL_{EVT} + SEL_{ARCH} + 0x1	Recall

The value of SEL_{ARCH} depends on the origin of the event:

- SEL_{ARCH} = $\text{SEL}_{\text{HST/ARCH}}$ (0x20) for events that occurred in the host.
- SEL_{ARCH} = $\text{SEL}_{\text{GST/ARCH}}$ (0x100) for events that occurred in the guest.

7.5 Architecture-Dependent Structures

7.5.1 Hypervisor Information Page

The architecture-dependent HIP structure is empty.

7.5.2 User Thread Control Block

-		IA32_KERNEL_GS_BASE			+0x1f0
IA32_FMASK		IA32_LSTAR			+0x1e0
IA32_STAR		IA32_EFER			+0x1d0
IA32_PAT		IA32_SYSENTER_EIP			+0x1c0
IA32_SYSENTER_ESP		IA32_SYSENTER_CS			+0x1b0
DR7		CR8			+0x1a0
CR4		CR3			+0x190
CR2		CR0			+0x180
PDPTE3		PDPTE2			+0x170
PDPTE1		PDPTE0			+0x160
Base IDTR		Limit IDTR	-		+0x150
Base GDTR		Limit GDTR	-		+0x140
Base LDTR		Limit LDTR	AR LDTR*	SEL LDTR	+0x130
Base TR		Limit TR	AR TR*	SEL TR	+0x120
Base GS		Limit GS	AR GS*	SEL GS	+0x110
Base FS		Limit FS	AR FS*	SEL FS	+0x100
Base ES		Limit ES	AR ES*	SEL ES	+0x0f0
Base DS		Limit DS	AR DS*	SEL DS	+0x0e0
Base SS		Limit SS	AR SS*	SEL SS	+0x0d0
Base CS		Limit CS	AR CS*	SEL CS	+0x0c0
Injection Error	Injection Info†	Activity	Interruptibility		+0x0b0
2nd Exit Qualification		1st Exit Qualification			+0x0a0
3rd Exec Controls	2nd Exec Controls	1st Exec Controls	Instruction Length		+0x090
RIP		RFLAGS			+0x080
R15		R14			+0x070
R13		R12			+0x060
R11		R10			+0x050
R9		R8			+0x040
R7 (RDI)		R6 (RSI)			+0x030
R5 (RBP)		R4 (RSP)			+0x020
R3 (RBX)		R2 (RDX)			+0x010
R1 (RCX)		R0 (RAX)			+0x000
48 32	16 0	48 32	16	0	

*See Section 7.5.2.1 for encoding details.
†See Section 7.5.2.2 for encoding details.

7.5.2.1 Encoding: Segment Access Rights

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

Field	Description
`U`	0 = Segment Usable 1 = Segment Unusable
`G`	Granularity
`D/B`	0 = 16-bit segment 1 = 32-bit segment
`L`	64-bit mode active (CS only)
`AVL`	Available for use by system software
`P`	Segment Present
`DPL`	Descriptor Privilege Level
`S`	0 = System 1 = Code or Data
`Type`	Segment Type

7.5.2.2 Encoding: Injection Information

V	~	N	I	E	Type	Vector
31		13	12	11	10 8	7 0

Field	Description
`V`	0 = Fields `E`, `Type`, `Vector` are invalid 1 = Fields `E`, `Type`, `Vector` are valid
`N`	0 = Do not request an NMI window 1 = Request an NMI window
`I`	0 = Do not request an interrupt window 1 = Request an interrupt window
`E`	0 = Do not deliver the error code from the UTCB `Injection Error` field 1 = Deliver the error code from the UTCB `Injection Error` field
`Type`	0 = External Interrupt 2 = Non-Maskable Interrupt 3 = Hardware Exception 4 = Software Interrupt 5 = Privileged Software Exception 6 = Software Exception 7 = Other Event (not delivered through IDT)
`Vector`	IDT Vector of Interrupt or Exception

7.5.3 Message Transfer Descriptor

The Message Transfer Descriptor (MTD), which controls the subset of the architectural state transferred during exceptions and intercepts, as described in Section 3.4.2, has the following layout:

FPU	-	KERNEL_GS	SYSCALL	EFER	PAT	SYSENTER	DR	CR	PDPTE	IDTR	GDTR	LDTR	TR	FS/GS	DS/ES	CS/SS	INJ	STA	QUAL	CTRL	RIP	RFLAGS	GPR_{8-15}	GPR_{0-7}	POISON
31		23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Each MTD bit controls the transfer of the listed architectural state to/from the respective fields in the UTCB (7.5.2) as follows:

- State with access `r` can be read from the architectural state into the UTCB.
- State with access `w` can be written from the UTCB into the architectural state.

MTD Bit	Access	Host Exception State	Guest Intercept State
POISON	w	Kills the EC	Kills the EC
GPR_{0-7}	rw	`R0 ... R7`	`R0 ... R7`
GPR_{8-15}	rw	`R8 ... R15`	`R8 ... R15`
RFLAGS	rw	`RFLAGS`*	`RFLAGS`
RIP	rw	`RIP`	`RIP, Instruction Length`
CTRL	w	-	`Execution Controls`
QUAL	r	`Exit Qualifications`†	`Exit Qualifications`
STA	rw	-	`Interruptibility State, Activity State`
INJ	rw	-	`Injection Info, Injection Error`
CS/SS	rw	-	`CS, SS (Selector, Base, Limit, AR)`
DS/ES	rw	-	`DS, ES (Selector, Base, Limit, AR)`
FS/GS	rw	-	`FS, GS (Selector, Base, Limit, AR)`
TR	rw	-	`TR (Selector, Base, Limit, AR)`
LDTR	rw	-	`LDTR (Selector, Base, Limit, AR)`
GDTR	rw	-	`GDTR (Base, Limit)`
IDTR	rw	-	`IDTR (Base, Limit)`
PDPTE	rw	-	`PDPTE0 ... PDPTE3`
CR	rw	-	`CR0, CR2, CR3, CR4, CR8`
DR	rw	-	`DR7`
SYSENTER	rw	-	`IA32_SYSENTER_{CS,ESP,EIP}`
PAT	rw	-	`IA32_PAT`
EFER	rw	-	`IA32_EFER`
SYSCALL	rw	-	`IA32_{STAR,LSTAR,FMASK}`
KERNEL_GS	rw	-	`IA32_KERNEL_GS_BASE`

*Only the arithmetic flags are writable.

†The 1st exit qualification contains the exception error code. The 2nd exit qualification contains the fault address.

7.6 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 [1, 3].

The hypercall identifier consists of the hypercall number and hypercall-specific flags, as illustrated in Figure 7.1.

flags	number
7 … 4	3 … 0

Figure 7.1: Hypercall Identifier

The status code returned from a hypercall has the format shown in Figure 7.2.

status
7 … 0

Figure 7.2: 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.

IPC Call

$pt_{[63-8]}$ $hypercall_{[7-0]}$	RDI	$\xrightarrow{\texttt{ipc\_call}}$	RDI	$status_{[7-0]}$
$mtd_{[31-0]}$	RSI		RSI	$mtd_{[31-0]}$
–	RCX		RCX	~
–	R11		R11	~
–	RIP	syscall	RIP	RIP+2

IPC Reply

$hypercall_{[7-0]}$	RDI	$\xrightarrow{\texttt{ipc\_reply}}$	RDI	pid
$mtd_{[31-0]}$	RSI		RSI	$mtd_{[31-0]}$
–	RCX		RCX	~
–	R11		R11	~
–	RIP	syscall	RIP	Portal IP

Create Protection Domain

$sel_{[63-8]}$ $hypercall_{[7-0]}$	RDI	$\xrightarrow{\texttt{create\_pd}}$	RDI	$status_{[7-0]}$
own	RSI		RSI	≡
–	RCX		RCX	~
–	R11		R11	~
–	RIP	syscall	RIP	RIP+2

Create Execution Context

		create_ec		
$sel_{[63-8]}$ $hypercall_{[7-0]}$	RDI	$\longrightarrow$	RDI	$status_{[7-0]}$
own	RSI		RSI	≡
$utcb_{[63-12]}$ $cpu_{[11-0]}$	RDX		RDX	≡
sp	RAX		RAX	≡
evt	R8		R8	≡
–	RCX		RCX	∼
–	R11		R11	∼
–	RIP	syscall	RIP	RIP+2

Create Scheduling Context

		create_sc		
$sel_{[63-8]}$ $hypercall_{[7-0]}$	RDI	$\longrightarrow$	RDI	$status_{[7-0]}$
own	RSI		RSI	≡
ec	RDX		RDX	≡
$quantum_{[31-12]}$ $prio_{[6-0]}$	RAX		RAX	≡
–	RCX		RCX	∼
–	R11		R11	∼
–	RIP	syscall	RIP	RIP+2

Create Portal

		create_pt		
$sel_{[63-8]}$ $hypercall_{[7-0]}$	RDI	$\longrightarrow$	RDI	$status_{[7-0]}$
own	RSI		RSI	≡
ec	RDX		RDX	≡
ip	RAX		RAX	≡
–	RCX		RCX	∼
–	R11		R11	∼
–	RIP	syscall	RIP	RIP+2

Create Semaphore

		create_sm		
$sel_{[63-8]}$ $hypercall_{[7-0]}$	RDI	$\longrightarrow$	RDI	$status_{[7-0]}$
own	RSI		RSI	≡
cnt	RDX		RDX	≡
–	RCX		RCX	∼
–	R11		R11	∼
–	RIP	syscall	RIP	RIP+2

Control Protection Domain

		ctrl_pd		
$spd_{[63-8]}$ $hypercall_{[7-0]}$	RDI	$\longrightarrow$	RDI	$status_{[7-0]}$
dpd	RSI		RSI	≡
$src_{[63-12]}$ $ord_{[6-2]}$ $spc_{[1-0]}$	RDX		RDX	≡
$dst_{[63-12]}$ $sh_{[11-10]}$ $ca_{[9-7]}$ $pmm_{[6-2]}$ $tbl_{[1-0]}$	RAX		RAX	≡
–	RCX		RCX	∼
–	R11		R11	∼
–	RIP	syscall	RIP	RIP+2

Control Execution Context

$ec_{[63-8]}$ $hypercall_{[7-0]}$	RDI	**ctrl_ec** $\longrightarrow$	RDI	$status_{[7-0]}$
–	RCX		RCX	∼
–	R11		R11	∼
–	RIP	syscall	RIP	RIP+2

Control Scheduling Context

$sc_{[63-8]}$ $hypercall_{[7-0]}$	RDI	**ctrl_sc** $\longrightarrow$	RDI	$status_{[7-0]}$
–	RSI		RSI	ticks
–	RCX		RCX	∼
–	R11		R11	∼
–	RIP	syscall	RIP	RIP+2

Control Portal

$pt_{[63-8]}$ $hypercall_{[7-0]}$	RDI	**ctrl_pt** $\longrightarrow$	RDI	$status_{[7-0]}$
pid	RSI		RSI	≡
$mtd_{[31-0]}$	RDX		RDX	≡
–	RCX		RCX	∼
–	R11		R11	∼
–	RIP	syscall	RIP	RIP+2

Control Semaphore

$sm_{[63-8]}$ $hypercall_{[7-0]}$	RDI	**ctrl_sm** $\longrightarrow$	RDI	$status_{[7-0]}$
ticks	RSI		RSI	≡
–	RCX		RCX	∼
–	R11		R11	∼
–	RIP	syscall	RIP	RIP+2

Control Hardware

$hypercall_{[7-0]}$	RDI	**ctrl_hw** $\longrightarrow$	RDI	$status_{[7-0]}$
p0	RSI		RSI	p0
p1	RDX		RDX	p1
p2	RAX		RAX	p2
p3	R8		R8	p3
–	RCX		RCX	∼
–	R11		R11	∼
–	RIP	syscall	RIP	RIP+2

Assign Interrupt

$sm_{[63-8]}$ $hypercall_{[7-0]}$	RDI	**assign_int** $\longrightarrow$	RDI	$status_{[7-0]}$
cpu	RSI		RSI	$msi\_addr_{[31-0]}$
$bdf_{[15-0]}$	RDX		RDX	$msi\_data_{[15-0]}$
–	RCX		RCX	∼
–	R11		R11	∼
–	RIP	syscall	RIP	RIP+2

Assign Device

$\text{pd}_{[63-8]}$ $\text{hypercall}_{[7-0]}$	RDI	$\xrightarrow{\texttt{assign\_dev}}$	RDI	$\text{status}_{[7-0]}$
$\text{smmu}_{[63-12]}$ $\text{tbl}_{[1-0]}$	RSI		RSI	$\equiv$
$\text{bdf}_{[15-0]}$	RDX		RDX	$\equiv$
–	RCX		RCX	$\sim$
–	R11		R11	$\sim$
–	RIP	syscall	RIP	RIP+2

Part V

Appendix

A Acronyms

ACPI	Advanced Configuration and Power Interface [10]
$\mathtt{ATTR}_{\mathtt{CA}}$	Cacheability Attribute (ARM, x86)
$\mathtt{ATTR}_{\mathtt{SH}}$	Shareability Attribute (ARM, x86)
BDF	PCI Bus:Device:Function
CAP	Capability
$\mathtt{CAP}_{\emptyset}$	Null Capability
$\mathtt{CAP}_{\mathtt{MEM}}$	Memory Capability
$\mathtt{CAP}_{\mathtt{MSR}}$	MSR Capability
$\mathtt{CAP}_{\mathtt{OBJ}}$	Object Capability
$\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{PD}}}$	PD Object Capability
$\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{EC}}}$	EC Object Capability
$\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{SC}}}$	SC Object Capability
$\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{PT}}}$	PT Object Capability
$\mathtt{CAP}_{\mathtt{OBJ}_{\mathtt{SM}}}$	SM Object Capability
$\mathtt{CAP}_{\mathtt{PIO}}$	I/O Port Capability
CPU	CPU Number
DMA	Direct Memory Access
EC	Execution Context
$\mathtt{EC}_{\mathtt{CURRENT}}$	Current Execution Context
$\mathtt{EC}_{\mathtt{ROOT}}$	Root Execution Context
ELF	Executable and Linkable Format [9]
FDT	Flattened Device Tree [4]
FPU	Floating Point Unit
HIP	Hypervisor Information Page
IP	Instruction Pointer
IPC	Inter-Process Communication
MSI	Message Signaled Interrupt [7]
MSR	Model-Specific Register
MTD	Message Transfer Descriptor
PCI	Peripheral Component Interconnect [7]
PD	Protection Domain
$\mathtt{PD}_{\mathtt{CURRENT}}$	Current Protection Domain
$\mathtt{PD}_{\mathtt{NOVA}}$	NOVA Protection Domain
$\mathtt{PD}_{\mathtt{ROOT}}$	Root Protection Domain
PID	Portal Identifier
PT	Portal
SC	Scheduling Context

$\mathtt{SC}_{\mathtt{CURRENT}}$	Current Scheduling Context
$\mathtt{SC}_{\mathtt{ROOT}}$	Root Scheduling Context
`SEL`	Capability Selector
$\mathtt{SEL}_{\mathtt{EVT}}$	Event Selector Base
$\mathtt{SEL}_{\mathtt{MEM}}$	Memory Capability Selector
$\mathtt{SEL}_{\mathtt{MSR}}$	MSR Capability Selector
$\mathtt{SEL}_{\mathtt{OBJ}}$	Object Capability Selector
$\mathtt{SEL}_{\mathtt{PIO}}$	I/O Port Capability Selector
`SID`	Stream Identifier
`SM`	Semaphore
`SMMU`	System Memory Management Unit
`SP`	Stack Pointer
$\mathtt{SPC}_{\mathtt{MEM}}$	Memory Space
$\mathtt{SPC}_{\mathtt{MSR}}$	MSR Space
$\mathtt{SPC}_{\mathtt{OBJ}}$	Object Space
$\mathtt{SPC}_{\mathtt{PIO}}$	I/O Port Space
$\mathtt{TYPE}_{\mathtt{SPC}}$	Space Type
$\mathtt{TYPE}_{\mathtt{TBL}}$	Table Type
`UEFI`	Unified Extensible Firmware Interface [11]
`UTCB`	User Thread Control Block
`VMM`	Virtual-Machine Monitor

`ipc_call`	Hypercall (ARM, x86): IPC Call
`ipc_reply`	Hypercall (ARM, x86): IPC Reply
`create_pd`	Hypercall (ARM, x86): Create Protection Domain
`create_ec`	Hypercall (ARM, x86): Create Execution Context
`create_sc`	Hypercall (ARM, x86): Create Scheduling Context
`create_pt`	Hypercall (ARM, x86): Create Portal
`create_sm`	Hypercall (ARM, x86): Create Semaphore
`ctrl_pd`	Hypercall (ARM, x86): Control Protection Domain
`ctrl_ec`	Hypercall (ARM, x86): Control Execution Context
`ctrl_sc`	Hypercall (ARM, x86): Control Scheduling Context
`ctrl_pt`	Hypercall (ARM, x86): Control Portal
`ctrl_sm`	Hypercall (ARM, x86): Control Semaphore
`ctrl_hw`	Hypercall (ARM, x86): Control Hardware
`assign_int`	Hypercall (ARM, x86): Assign Interrupt
`assign_dev`	Hypercall (ARM, x86): Assign Device

B Bibliography

[1] *AMD64 Architecture Programmer's Manual Volume 2: System Programming*. Advanced Micro Devices. URL https://developer.amd.com/resources/developer-guides-manuals. Document Number: 24593. 47, 53

[2] *ARM Architecture Reference Manual ARMv8, for ARMv8-A Architecture Profile*. ARM Limited. URL https://developer.arm.com/docs/ddi0487/latest. Document Number: DDI0487. 38, 39, 43

[3] *Intel® 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4*. Intel Corporation. URL https://software.intel.com/en-us/articles/intel-sdm. Document Number: 325462. 47, 49, 53

[4] *Devicetree Specification*. Linaro Limited, 2020. URL https://www.devicetree.org/specifications. Version 0.3. 36, 38, 58

[5] Yoshinori K. Okuji, Bryan Ford, Erich Stefan Boleyn, and Kunihiro Ishiguro. *The Multiboot Specification*, 2010. URL https://www.gnu.org/software/grub/manual/multiboot/multiboot.pdf. Version 0.6.96. 36, 46, 47

[6] Yoshinori K. Okuji, Bryan Ford, Erich Stefan Boleyn, Kunihiro Ishiguro, Vladimir Serbinenko, and Daniel Kiper. *The Multiboot2 Specification*, 2016. URL https://www.gnu.org/software/grub/manual/multiboot2/multiboot.pdf. Version 2.0. 36, 46, 47

[7] *PCI Local Bus Specification*. PCI-SIG, 2004. URL https://pcisig.com/specifications. Revision 3.0. 58

[8] 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. ISBN 978-1-60558-577-2. URL https://doi.acm.org/10.1145/1755913.1755935. 2

[9] *Executable and Linking Format (ELF) Specification*. TIS Committee, 1995. URL https://refspecs.linuxbase.org/elf/elf.pdf. Version 1.2. 58

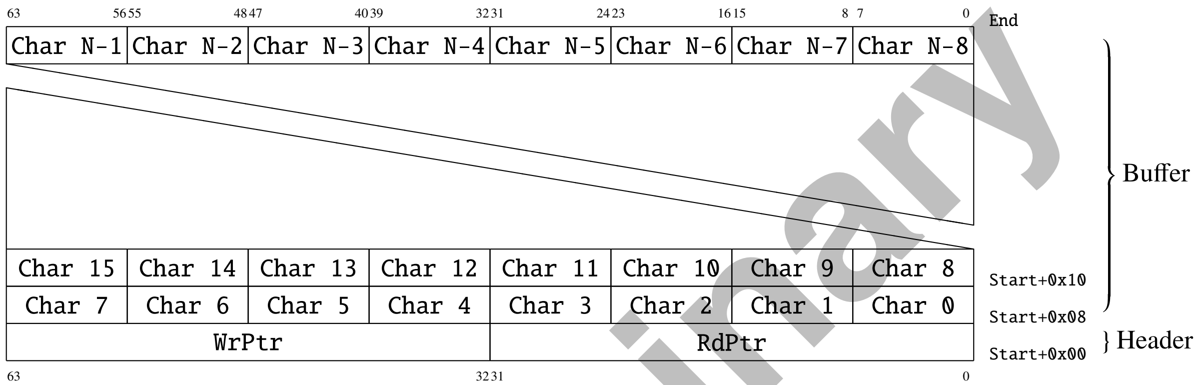
[10] *Advanced Configuration and Power Interface (ACPI) Specification*. UEFI Forum, Inc., 2019. URL https://uefi.org/specifications. Version 6.3. 58

[11] *Unified Extensible Firmware Interface (UEFI) Specification*. UEFI Forum, Inc., 2019. URL https://uefi.org/specifications. Version 2.8. 38, 47, 59

C Console

C.1 Memory-Buffer Console

The NOVA microhypervisor implements a memory-buffer console that provides run-time debug output. The memory-buffer console consists of a signaling semaphore (see 5.1.2) and an in-memory data structure with a header and a buffer as follows:



The start address and end address of the memory-buffer console are conveyed in the HIP.

The buffer size (`N` characters) can be computed as:

```
N = MBUF End Address - MBUF Start Address - MBUF Header Size
```

The fields of the header are used as follows:

- `RdPtr` ranges from `0` . . . `N-1`.
  It points to the **next** character in the buffer that the console consumer will read and is typically advanced by the console consumer.
- `WrPtr` ranges from `0` . . . `N-1`.
  It points to the **next** character in the buffer that the NOVA microhypervisor will write and is only advanced by the NOVA microhypervisor.
- The buffer is empty if `RdPtr` is equal to `WrPtr`.
- Otherwise `WrPtr` is ahead of `RdPtr`, wrapping around the buffer size `N` accordingly, i.e. character `N+x` will be stored in the same buffer slot as character `x`.
- If the buffer becomes full, the NOVA microhypervisor advances `RdPtr`, forcing the oldest character to be discarded from the buffer.
- At the end of each line, the NOVA microhypervisor invokes `ctrl_sm` (`Up`) on the signaling semaphore. The console consumer should use `ctrl_sm` (`Down`) on the signaling semaphore instead of polling `WrPtr`.

C.2 UART Console

Additionally several different UART consoles can be used to provide boot-time-only debug output of the microhypervisor. UART consoles should be configured for `115200 baud` and `8N1` mode.

D Download

The source code of the NOVA microhypervisor and the latest version of this document can be downloaded from GitHub.

https://github.com/udosteinberg/NOVA