NOVA Microhypervisor Interface Specification

Udo Steinberg
udo@hypervisor.org

May 19, 2022

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

Notation

The key words **must**, **must not**, **required**, **should**, **should not**, **recommended**, **may** and **optional** in this document are to be interpreted as described in RFC 2119 [1].

Throughout this document, the following symbols are used:

$\sim$ 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.

$\equiv$ 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 **NOVA OS Virtualization Architecture** [2] (NOVA) facilitates the coexistence of multiple legacy guest operating systems and a user-mode host framework on a single platform. The core system leverages hardware virtualization technology provided by modern x86 or Arm platforms and comprises the NOVA microhypervisor and one or more Virtual-Machine Monitors (VMMs).

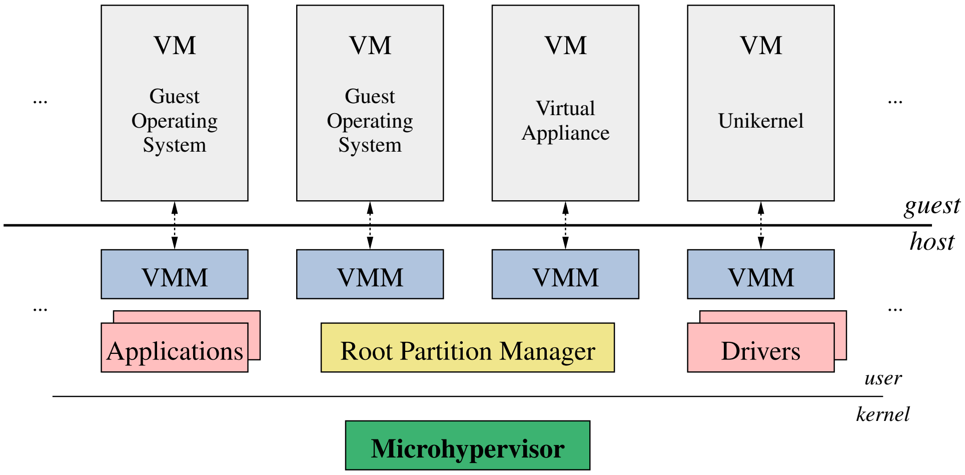


Figure 1.1: System Architecture

Figure 1.1 shows the structure of the system. The microhypervisor is the only component executing in privileged host/kernel mode. It isolates the various user-mode components, including the virtual-machine monitors, from one another by placing them in different protection domains in unprivileged host/user mode. Each legacy guest operating system runs in its own virtual-machine environment in guest mode and is therefore isolated from the other components.

Besides spatial and temporal 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 channels and signaling between different protection domains.

The virtual-machine monitors handle virtualization events and implement virtual devices that enable legacy guest operating systems to function in the same manner as they would on bare-metal hardware. Providing this functionality outside the microhypervisor in the VMMs reduces the size of the trusted computing base significantly for all components that do not require virtualization support.

The architecture and interfaces of the VMM and the user-mode host 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 spatial isolation.
2. Access to a Protection Domain is controlled by a PD Object Capability (CAP$_{\text{OBJ}_{\text{PD}}}$).
3. A Protection Domain is composed of a set of spaces that store Capabilities (CAPs) to kernel objects or platform resources that can be accessed by Execution Contexts (ECs) within that PD. Not all spaces are available on all architectures (see 5.1.3 for details). The following subsections detail all spaces.

2.1.1 Object Space

1. An Object Capability Selector (SEL$_{\text{OBJ}}$) serves as index into the Object Space and selects a slot.
2. Each slot of the Object Space contains either a Null Capability (CAP$_0$) or an Object Capability (CAP$_{\text{OBJ}}$) that refers to a kernel object.
3. Each hypercall issued from within the PD explicitly specifies the SEL$_{\text{OBJ}}$ to select the CAP$_{\text{OBJ}}$ for the kernel object on which it operates.

2.1.2 Memory Space

1. A Memory Capability Selector (SEL$_{\text{MEM}}$) serves as index into the Memory Space and selects a slot.
2. Each slot of the Memory Space contains either a Null Capability (CAP$_0$) or a Memory Capability (CAP$_{\text{MEM}}$) that refers to a 4 KiB page frame in physical memory.
3. Each memory access issued from within the PD implicitly uses the virtual page number (`VirtAddr >> 12`) of the access as SEL$_{\text{MEM}}$ to select the CAP$_{\text{MEM}}$ for the 4 KiB page frame on which it operates.

2.1.3 I/O Port Space

1. An I/O Port Capability Selector (SEL$_{\text{PIO}}$) serves as index into the I/O Port Space and selects a slot.
2. Each slot of the I/O Port Space contains either a Null Capability (CAP$_0$) or an I/O Port Capability (CAP$_{\text{PIO}}$) that refers to the physical I/O port corresponding to the slot number.
3. Each I/O access (`IN`/`OUT` instruction) issued from within the PD implicitly uses the I/O port number of the access as SEL$_{\text{PIO}}$ to select the CAP$_{\text{PIO}}$ for the I/O port on which it operates.

2.1.4 MSR Space

1. An MSR Capability Selector (SEL$_{\text{MSR}}$) serves as index into the MSR Space and selects a slot.
2. Each slot of the MSR Space contains either a Null Capability (CAP$_0$) or an MSR Capability (CAP$_{\text{MSR}}$) that refers to the physical MSR corresponding to the slot number.
3. Each MSR access (`RDMSR`/`WRMSR` instruction) issued from within the PD implicitly uses the MSR number of the access as SEL$_{\text{MSR}}$ to select the CAP$_{\text{MSR}}$ for the MSR on which it operates.

2.2 Execution Context

1. The Execution Context (EC) is an abstraction for an activity within a PD.
2. Access to an Execution Context is controlled by an EC Object Capability ($\text{CAP}_{\text{OBJ}_{\text{EC}}}$).
3. An EC is permanently bound to exactly one physical CPU.
4. An EC is permanently bound to the PD for which it was created.
5. There exist three types of Execution Context:
   - Local Threads – these may have PTs (but no SCs) bound to it.
   - Global Threads – these may have an SC (but no PTs) bound to it.
   - Virtual CPUs – these may have an SC (but no PTs) bound to it.
6. An EC comprises the following state:
   - Reference to bound PD (2.1)
   - Event Selector Base (SEL_{EVT})
   - User Thread Control Block (UTCB) (4.3)
   - 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 prioritization and temporal isolation.
2. Access to a Scheduling Context is controlled by an SC Object Capability ($\text{CAP}_{\text{OBJ}_{\text{SC}}}$).
3. An SC is permanently bound to exactly one physical CPU.
4. An SC is permanently bound to the EC for which it was created.
5. Donation allows another EC to consume the budget of the SC for the duration of the donation.
6. A scheduling context comprises the following state:
   - Reference to bound EC (2.2)
   - Scheduling priority – numerically higher priorities always preempt numerically lower priorities
   - Scheduling budget – time after which the SC can be preempted by an SC with the same priority

2.4 Portal

1. A Portal (PT) represents a dedicated entry point into the PD for which the portal was created.
2. Access to a Portal is controlled by a PT Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PT}}}$).
3. A PT is permanently bound to the EC for which it was created.
4. A portal comprises the following state:
   - Reference to bound EC (2.2)
   - Message Transfer Descriptor (MTD) (4.4)
   - Entry Instruction Pointer (IP)
   - 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 (ECs).
2. Access to a Semaphore is controlled by a SM Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{SM}}}$).

3 Hardware Resources

3.1 System Time Counter

The system time is represented by an unsigned 64-bit System Time Counter (STC) with the following properties:

1. The STC starts with a power-on value of 0.
2. Subsequent reads of the STC return a higher value that reflects the platform uptime.
3. While the platform is in a shallow sleep state, the STC retains its current value.
4. While the platform is running, the STC monotonically increments at a fixed frequency, which is conveyed in the Hypervisor Information Page (HIP).
5. The STC and its frequency are synchronized across all CPUs. Applications can use both values to convert between system time and wall clock time.
6. Applications can obtain the current STC value as follows:

   Arm: By reading `CNTVCT_EL0` via the `MRS` instruction [3].

   x86: By reading `IA32_TSC` via the `RDTSC` instruction [4, 5].

Part III

Application Programming Interface

4 Data Types

4.1 Capability

A Capability (CAP) is a reference to a resource coupled with 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).

4.1.1 Null Capability

A Null Capability (CAP_0) does not refer to anything and carries no permissions.

4.1.2 Object Capability

An Object Capability (CAP_{OBJ}) is stored in the Object Space (SPC_{OBJ}) of a PD and refers to a kernel object.

4.1.2.1 PD Object Capability

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

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

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.

4.1.2.2 EC Object Capability

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

~	BIND_{SC}	BIND_{PT}	~	CTRL
4	3	2	1	0

CTRL	`ctrl_ec` permitted if set.
BIND_{PT}	`create_pt` can bind a Portal (PT) to the EC if set.
BIND_{SC}	`create_sc` can bind a Scheduling Context (SC) to the EC if set.

4.1.2.3 SC Object Capability

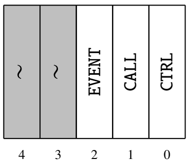
An SC Object Capability ($\text{CAP}_{\text{OBJ}_{\text{SC}}}$) refers to a Scheduling Context (SC) and carries the following permissions:

~	~	~	~	CTRL
4	3	2	1	0

CTRL	`ctrl_sc` permitted if set.

4.1.2.4 PT Object Capability

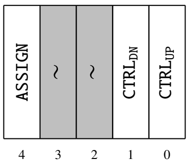
A PT Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PT}}}$) refers to a Portal (PT) and carries the following permissions:



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

4.1.2.5 SM Object Capability

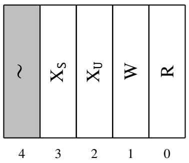
An SM Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{SM}}}$) refers to a Semaphore (SM) and carries the following permissions:



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

4.1.3 Memory Capability

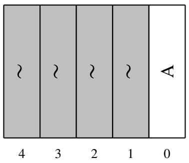
A Memory Capability ($\mathrm{CAP}_{\mathrm{MEM}}$) is stored in the Memory Space ($\mathrm{SPC}_{\mathrm{MEM}}$) of a PD, refers to a 4 KiB page frame, and carries the following permissions:



R	the page frame is readable if set.
W	the page frame is writable if set.
$\mathtt{X}_{\mathrm{U}}$ [‡]	the page frame is executable (in user mode) if set.
$\mathtt{X}_{\mathrm{S}}$ [‡]	the page frame is executable (in supervisor mode) if set.

4.1.4 I/O Port Capability

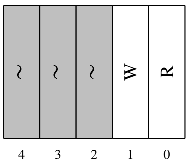
A I/O Port Capability ($\mathrm{CAP}_{\mathrm{PIO}}$) is stored in the I/O Port Space ($\mathrm{SPC}_{\mathrm{PIO}}$) of a PD, refers to an I/O port, and carries the following permissions:



A	the I/O port is accessible (via `IN`/`OUT`) if set.

4.1.5 MSR Capability

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



R	the MSR is readable (via `RDMSR`) if set.
W	the MSR is writable (via `WRMSR`) if set.

[†] This permission bit is only defined for interrupt semaphores.

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

4.2 Capability Selector

A Capability Selector (SEL) is an application-visible unsigned number as follows:

- An Object Capability Selector (SEL_{OBJ}) indexes into the Object Space (SPC_{OBJ}) of a Protection Domain (PD) and selects a slot that contains either a Null Capability (CAP_0) or an Object Capability (CAP_{OBJ}).
- A Memory Capability Selector (SEL_{MEM}) indexes into the Memory Space (SPC_{MEM}) of a Protection Domain (PD) and selects a slot that contains either a Null Capability (CAP_0) or a Memory Capability (CAP_{MEM}).
- An I/O Port Capability Selector (SEL_{PIO}) indexes into the I/O Port Space (SPC_{PIO}) of a Protection Domain (PD) and selects a slot that contains either a Null Capability (CAP_0) or an I/O Port Capability (CAP_{PIO}).
- An MSR Capability Selector (SEL_{MSR}) indexes into the MSR Space (SPC_{MSR}) of a Protection Domain (PD) and selects a slot that contains either a Null Capability (CAP_0) or an MSR Capability (CAP_{MSR}).

4.3 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 has a size of one memory page (4 KiB). Because a UTCB is owned by the microhypervisor, it cannot be delegated using `ctrl_pd`.

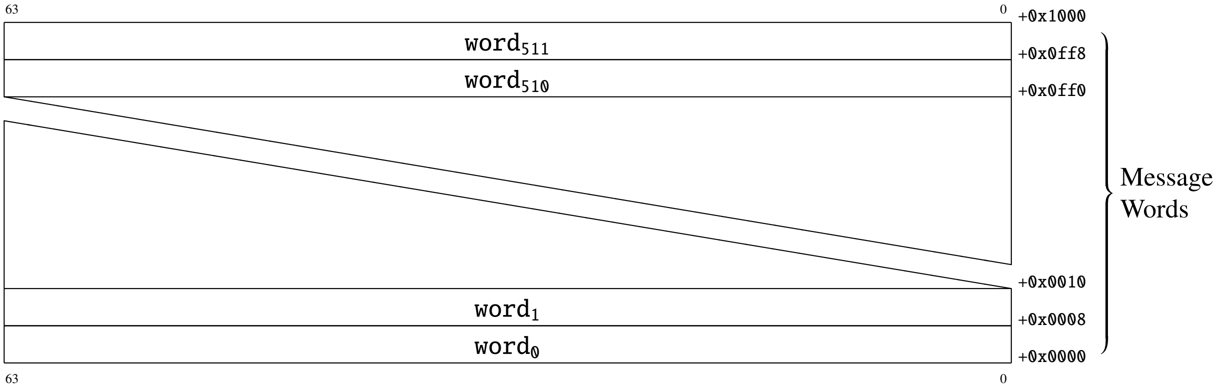
To ensure proper visibility of loads and stores with relaxed memory ordering, application programs are expected to access a UTCB only from the EC to which that UTCB is bound.

4.3.1 Regular Layout

During regular IPC (see 4.4.1), the UTCB is used for data transfer.

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 with 512 message words (see below).
- The data is copied from low words to high words, beginning with $\texttt{word}_0$.
- 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.



4.3.2 Architectural Layout

During architectural IPC (see 4.4.2), the UTCB is used for state transfer.

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 affected EC and callee EC execute.
- The state transfer uses the architectural layout (Arm, x86).
- 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.

4.4 Message Transfer Descriptor

4.4.1 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 4.3.1) as shown in Figure 4.1:

- 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).

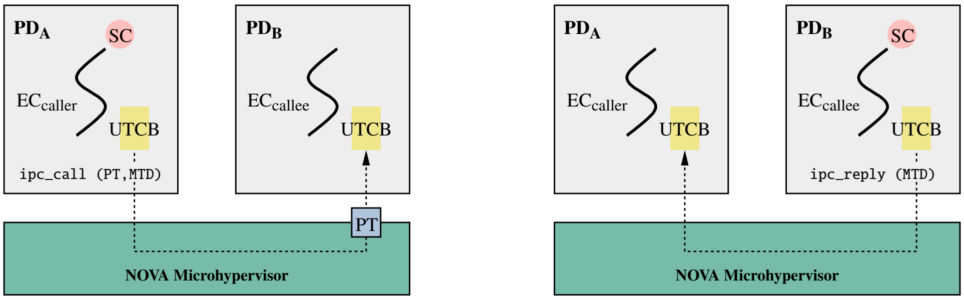


Figure 4.1: Regular IPC

4.4.2 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 4.3.2) as shown in Figure 4.2:

- During an exception/intercept, it specifies the subset of registers to transfer from the architectural state of the affected 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 affected EC (receiver).

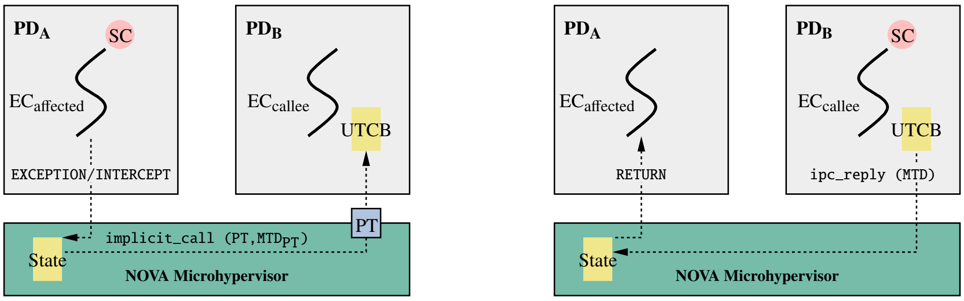


Figure 4.2: Architectural IPC

4.5 Scheduling Context Descriptor

The Scheduling Context Descriptor (SCD) describes the configuration of a Scheduling Context (SC).

-	COS	Prio	Budget
63 39	38 23	22 16	15 0

The fields are defined as follows:

Budget

Specifies the scheduling budget in milliseconds – must be > 0.

Prio

Specifies the scheduling priority – must be > 0.

COS

Specifies the Class Of Service – valid values depend on architectural COS support (Arm, x86):

- If COS is not supported, then this field must be 0.
- If COS is supported, then this field must be $< \text{COS}_{\text{NUM}}$.

5 Hypercalls

5.1 Definitions

5.1.1 Hypercall Numbers

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

Number	Hypercall	Section
0x0	ipc_call	5.2.1
0x1	ipc_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	ctrl_pd	5.4.1
0x8	ctrl_ec	5.4.2
0x9	ctrl_sc	5.4.3
0xa	ctrl_pt	5.4.4
0xb	ctrl_sm	5.4.5
0xc	ctrl_hw	5.5.1
0xd	assign_int	5.5.2
0xe	assign_dev	5.5.3
0xf	*reserved for future use*	

5.1.2 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
≥0xb	*reserved for future use*	

5.1.3 Space Type

The following table lists the currently defined space types and for which architectures they are valid (✓):

Number	TYPE_{SPC}	Arm	x86	Description
`0x0`	SPC_{OBJ}	✓	✓	Object Space
`0x1`	SPC_{MEM}	✓	✓	Memory Space
`0x2`	SPC_{PIO}	×	✓	I/O Port Space
`0x3`	SPC_{MSR}	×	✓	MSR Space
≥`0x4`	*reserved for future use*			

5.1.4 Access Type

The following table lists the currently defined access types and for which space types they are valid (✓):

Number	TYPE_{ACC}	SPC_{OBJ}	SPC_{MEM}	SPC_{PIO}	SPC_{MSR}	Description
`0x0`	`CPU_HST`	✓	✓	✓	×	CPU Access from Host
`0x1`	`CPU_GST`	×	✓	✓	✓	CPU Access from Guest
`0x2`	`DMA_HST`	×	✓	×	×	DMA Access from Host
`0x3`	`DMA_GST`	×	✓	×	×	DMA Access from Guest
≥`0x4`	*reserved for future use*					

5.2 Communication

5.2.1 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$_{\text{CURRENT}}$ (caller) to the EC (callee) to which the specified Portal (PT) is bound.

Prior to the hypercall:

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

If the hypercall completed successfully:

- If **T=0 (No Timeout)**: If the callee EC was still busy handling a prior `ipc_call`, then the caller EC has helped run that prior `ipc_call` 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 `mtd` parameter have been updated by the reply message.
- The Current Scheduling Context (SC$_{\text{CURRENT}}$) 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$_{\text{CURRENT}}$.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

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

BAD_CPU

- Caller EC and callee EC are on different CPUs.

TIMEOUT

- If **T=1 (Timeout)**: The callee EC is still busy handling a prior `ipc_call`.

ABORTED

- The callee EC is dead and the operation aborted.

5.2.2 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 `mtd` parameter have been updated by the incoming message.
- Execution of the callee EC continues at the Instruction Pointer (IP) configured in the called PT.

5.3 Object Creation

5.3.1 Create Protection Domain

Parameters:

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

Flags:

0	0	0	0
3	2	1	0

Description:

Creates a new Protection Domain (PD).

Prior to the hypercall:

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `sel` } must refer to a Null Capability (CAP_0).
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `own` } must refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) with permission `PD`.

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 with defined permissions inherited from { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `own` }.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `sel` } did not refer to a Null Capability (CAP_0).
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `own` } did not refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) or that capability had insufficient permissions.

INS_MEM

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

5.3.2 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	T
3	2	1	0

Description:

Creates a new Execution Context (EC).

Prior to the hypercall:

- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `sel` } must refer to a Null Capability (CAP_0).
- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `own` } must refer to a PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) with permission `EC`.

If the hypercall completed successfully:

- If **V=0 (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`.
  - If **T=0 (Local Thread)**: Portals (PTs) may subsequently be bound to that EC and the EC will run whenever any of those bound portals is called.
  - If **T=1 (Global Thread)**: 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` and `sp` were ignored.
  - If **T=0**: The virtual CPU uses no time adjustment.
  - If **T=1**: The virtual CPU uses time offsetting.
- The created EC will be able to use FPU instructions only if **F=1 (FPU)**. Otherwise any FPU access by that EC will generate an exception.
- The created EC is bound to the PD referred to by { $\text{PD}_{\text{CURRENT}}$, 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 { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `own` }.
- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `sel` } refers to an EC Object Capability ($\text{CAP}_{\text{OBJ}_{\text{EC}}}$) for the created EC with all defined permissions set.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `sel` } did not refer to a Null Capability (CAP_0).
- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `own` } did not refer to a PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) or that capability had insufficient permissions.

BAD_CPU

- The CPU number is invalid.

BAD_FTR

- Virtual CPUs are not supported by the platform.

BAD_PAR

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

INS_MEM

- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `own` } had insufficient memory resources for EC creation.

5.3.3 Create Scheduling Context

Parameters:

```
status = create_sc (SELOBJ sel,        // Created SC
                    SELOBJ own,        // Owner PD
                    SELOBJ ec,         // Bound EC
                    SCD    scd);       // Scheduling Context Descriptor
```

Flags:

0	0	0	0
3	2	1	0

Description:

Creates a new Scheduling Context (SC).

Prior to the hypercall:

- { PD$_{CURRENT}$, SEL$_{OBJ}$ `sel` } must refer to a Null Capability (CAP$_0$).
- { PD$_{CURRENT}$, SEL$_{OBJ}$ `own` } must refer to a PD Object Capability (CAP$_{OBJ_{PD}}$) with permission SC.
- { PD$_{CURRENT}$, SEL$_{OBJ}$ `ec` } must refer to an EC Object Capability (CAP$_{OBJ_{EC}}$) with permission BIND$_{SC}$.

If the hypercall completed successfully:

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

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { PD$_{CURRENT}$, SEL$_{OBJ}$ `sel` } did not refer to a Null Capability (CAP$_0$).
- { 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$_{OBJ}$ `ec` } did not refer to an EC Object Capability (CAP$_{OBJ_{EC}}$) or that capability had insufficient permissions.
- Binding the SC to the EC failed, e.g. because the EC is a local EC.

BAD_PAR

- At least one SCD field in `scd` was invalid.

INS_MEM

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

5.3.4 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:

- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `sel` } must refer to a Null Capability (CAP_0).
- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `own` } must refer to a PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) with permission `PT`.
- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `ec` } must refer to an EC Object Capability ($\text{CAP}_{\text{OBJ}_{\text{EC}}}$) with permission $\texttt{BIND}_{\texttt{PT}}$.

If the hypercall completed successfully:

- A new Portal (PT) has been created.
- The created PT is bound to the EC referred to by { $\text{PD}_{\text{CURRENT}}$, SEL_{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 { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `own` }.
- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `sel` } refers to an PT Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PT}}}$) for the created PT with all defined permissions set.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `sel` } did not refer to a Null Capability (CAP_0).
- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `own` } did not refer to a PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) or that capability had insufficient permissions.
- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `ec` } did not refer to an EC Object Capability ($\text{CAP}_{\text{OBJ}_{\text{EC}}}$) or that capability had insufficient permissions.
- Binding the PT to the EC failed, e.g. because the EC is not a local EC.

INS_MEM

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

5.3.5 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:

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `sel` } must refer to a Null Capability (CAP_0).
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `own` } must refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) with permission `SM`.

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 with all defined permissions set.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `sel` } did not refer to a Null Capability (CAP_0).
- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `own` } did not refer to a PD Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PD}}}$) or that capability had insufficient permissions.

INS_MEM

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

5.4 Object Control

5.4.1 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_ACC  acc,       // Access 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:

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

If the hypercall completed successfully:

- If **spc=SPC$_{\text{OBJ}}$**: All CAP$_{\text{OBJ}}$ and CAP$_0$ from source SEL range { PD spd, SEL$_{\text{OBJ}}$ src...src+2$^{\texttt{ord}}$-1 } were delegated to destination SEL range { PD dpd, SEL$_{\text{OBJ}}$ dst...dst+2$^{\texttt{ord}}$-1 }. Any pre-existing CAP$_{\text{OBJ}}$ in the destination selector range were revoked. The parameters `acc`, `ca` and `sh` were ignored.
- If **spc=SPC$_{\text{MEM}}$**: All CAP$_{\text{MEM}}$ and CAP$_0$ from source SEL range { PD spd, SEL$_{\text{MEM}}$ src...src+2$^{\texttt{ord}}$-1 } were delegated to destination SEL range { PD dpd, SEL$_{\text{MEM}}$ dst...dst+2$^{\texttt{ord}}$-1 }. Any pre-existing CAP$_{\text{MEM}}$ in the destination selector range were revoked.

  Delegation of Physical Memory:
  If `spd` refers to a PD Object Capability (CAP$_{\text{OBJ}_{\text{PD}}}$) for PD$_{\text{NOVA}}$, then the source selectors are physical page numbers (see 6.1.2) and the cacheability and shareability attribute of each destination capability were *set* to `ca` and `sh` respectively.

  Delegation of Virtual Memory:
  If `spd` refers to a PD Object Capability (CAP$_{\text{OBJ}_{\text{PD}}}$) for any other PD, then the source selectors are virtual page numbers and the cacheability and shareability attribute of each destination capability were *inherited* from the respective source capability, i.e. the parameters `ca` and `sh` were ignored.
- If **spc=SPC$_{\text{PIO}}$**: All CAP$_{\text{PIO}}$ and CAP$_0$ from source SEL range { PD spd, SEL$_{\text{PIO}}$ src...src+2$^{\texttt{ord}}$-1 } were delegated to destination SEL range { PD dpd, SEL$_{\text{PIO}}$ dst...dst+2$^{\texttt{ord}}$-1 }. Any pre-existing CAP$_{\text{PIO}}$ in the destination selector range were revoked. The parameters `ca` and `sh` were ignored.
- If **spc=SPC$_{\text{MSR}}$**: All CAP$_{\text{MSR}}$ and CAP$_0$ from source SEL range { PD spd, SEL$_{\text{MSR}}$ src...src+2$^{\texttt{ord}}$-1 } were delegated to destination SEL range { PD dpd, SEL$_{\text{MSR}}$ dst...dst+2$^{\texttt{ord}}$-1 }. Any pre-existing CAP$_{\text{MSR}}$ in the destination selector range were revoked. The parameters `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 (CAP_0) or if the destination capability has zero permissions after masking, then the destination capability is now a Null Capability (CAP_0).
- 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

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

BAD_PAR

- SEL src or SEL dst was not order-aligned.
- SEL $\text{src}+2^{\text{ord}}-1$ or SEL $\text{dst}+2^{\text{ord}}-1$ was larger than the maximum selector number.
- If **spc=SPC_{PIO} or spc=SPC_{MSR}**: SEL src was not equal to SEL dst.
- TYPE_{SPC} spc or TYPE_{ACC} acc was not valid, i.e. not supported by the architecture.
- ATTR_{CA} ca or ATTR_{SH} sh was not valid, i.e. not supported by the architecture.

INS_MEM

- { $\text{PD}_{\text{CURRENT}}$, SEL_{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 all destination capabilities up to the first allocation failure have been granted.

5.4.2 Control Execution Context

Parameters:

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

Flags:

0	0	0	S
3	2	1	0

Description:

Prior to the hypercall:

- { PD$_{\text{CURRENT}}$, SEL$_{\text{OBJ}}$ `ec` } must refer to an EC Object Capability (CAP$_{\text{OBJ}_{\text{EC}}}$) with permission `CTRL`.

If the hypercall completed successfully:

- The EC referred to by { PD$_{\text{CURRENT}}$, SEL$_{\text{OBJ}}$ `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 Event Portal (Arm, x86).
- If **S=0 (Weak Recall)**:
  - 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 Recall)**:
  - 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$_{\text{CURRENT}}$, SEL$_{\text{OBJ}}$ `ec` } did not refer to an EC Object Capability (CAP$_{\text{OBJ}_{\text{EC}}}$) or that capability had insufficient permissions.

5.4.3 Control Scheduling Context

Parameters:

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

Flags:

0	0	0	0
3	2	1	0

Description:

Prior to the hypercall:

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

If the hypercall completed successfully:

- The microhypervisor has returned the total consumed execution time as System Time Counter (STC) value for the SC referred to by { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} sc }.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

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

5.4.4 Control Portal

Parameters:

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

Flags:

0	0	0	0
3	2	1	0

Description:

Prior to the hypercall:

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `pt` } must refer to a PT Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{PT}}}$) 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 { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `pt` }.
- Subsequent portal traversals will use the new MTD and return the new PID.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

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

5.4.5 Control Semaphore

Parameters:

```
status = ctrl_sm (SEL_OBJ  sm,              // Semaphore
                  UINT     stc);            // Absolute Timeout
```

Flags:

0	0	Z	D
3	2	1	0

Description:

Prior to the hypercall:

- If **D=0** (**Semaphore Up**):
  - { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} sm } must refer to an SM Object Capability ($\text{CAP}_{\text{OBJ}_{\text{SM}}}$) with permission CTRL_{UP}.
- If **D=1** (**Semaphore Down**):
  - { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} sm } must refer to an SM Object Capability ($\text{CAP}_{\text{OBJ}_{\text{SM}}}$) with permission CTRL_{DN}.

If the hypercall completed successfully:

- If **D=0** (**Semaphore 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 timeout value and the Z-flag were ignored.
- If **D=1** (**Semaphore 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 $\text{EC}_{\text{CURRENT}}$ on the semaphore. If the timeout value was non-zero, $\text{EC}_{\text{CURRENT}}$ unblocks with a timeout status when the System Time Counter (STC) reaches or exceeds the specified 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

- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} sm } did not refer to an SM Object Capability ($\text{CAP}_{\text{OBJ}_{\text{SM}}}$) 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`.

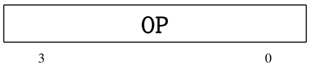
5.5 Platform Management

5.5.1 Control Hardware

Parameters:

```
status = ctrl_hw (UINT desc);          // Descriptor
```

Flags:

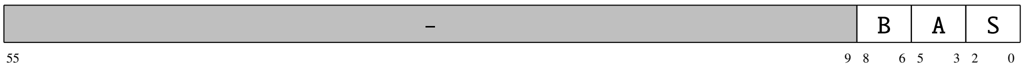


Description:

Modifies the platform hardware configuration or power management state.

Prior to the hypercall:

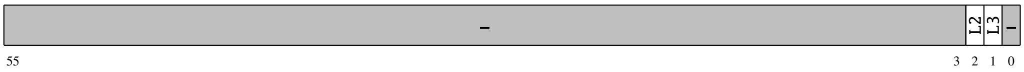
- $\text{PD}_{\text{CURRENT}}$ must be the Root Protection Domain (PD_{ROOT}).
- If **OP=0 (S-State Transition)**:
  - The descriptor `desc` uses the following encoding:



  - The value `S` designates the platform-wide sleep or reset state that shall be entered. The values `A` and `B` are the first two bytes of the respective `\_Sx` package in the ACPI root namespace as follows:

S	A	B	Shallow	Description
`0x1`	`\_S1[0]`	`\_S1[1]`	✓	S1: Power-On Suspend
`0x2`	`\_S2[0]`	`\_S2[1]`	✓	S2: Standby
`0x3`	`\_S3[0]`	`\_S3[1]`	✓	S3: Suspend to RAM
`0x4`	`\_S4[0]`	`\_S4[1]`	×	S4: Suspend to Disk
`0x5`	`\_S5[0]`	`\_S5[1]`	×	S5: Soft Off
`0x7`	`0x0`	`0x0`	×	Platform Reset

  - The caller is responsible for invoking the necessary pre-sleep ACPI methods, for transitioning platform devices into a suitable Dx sleep state, and for programming wakeup events.
- If **OP=4 (QOS Configuration)**:
  - The descriptor `desc` uses the following encoding:



  - Only Code and Data Prioritization (CDP) settings supported by the ambient CPU are valid.
    - ∗ The L3 bit disables (0) or enables (1) CDP_{L3} on the ambient CPU.
    - ∗ The L2 bit disables (0) or enables (1) CDP_{L2} on the ambient CPU.
  - CAT/CDP or MBA settings cannot be configured for a CPU until a valid QOS configuration has been established for that CPU. Subsequently, that QOS configuration cannot be changed anymore.

- If **OP=5 (CAT/CDP L3 Capacity Bitmask)**:
  - The descriptor `desc` uses the following encoding:

-	`L3 Capacity Bitmask`	`N`
55 48	47 16	15 0

  - N designates the CPU-local Class Of Service (COS) that shall be configured. Only COS below COS_{L3} of the ambient CPU are valid.
    - * If CDP_{L3} is disabled on the ambient CPU:
      - · To configure the CAT_{L3} Capacity Bitmask for a COS, use `N=COS`.
    - * If CDP_{L3} is enabled on the ambient CPU:
      - · To configure the $\text{CDP}_{\text{L3-Data}}$ Capacity Bitmask for a COS, use `N=(COS<<1)`.
      - · To configure the $\text{CDP}_{\text{L3-Code}}$ Capacity Bitmask for a COS, use `N=(COS<<1)+1`.
  - For the L3 Capacity Bitmask, all (and only) contiguous combinations of 1-bits up to the highest capacity bit supported by the ambient CPU are valid.
- If **OP=6 (CAT/CDP L2 Capacity Bitmask)**:
  - The descriptor `desc` uses the following encoding:

-	`L2 Capacity Bitmask`	`N`
55 48	47 16	15 0

  - N designates the CPU-local Class Of Service (COS) that shall be configured. Only COS below COS_{L2} of the ambient CPU are valid.
    - * If CDP_{L2} is disabled on the ambient CPU:
      - · To configure the CAT_{L2} Capacity Bitmask for a COS, use `N=COS`.
    - * If CDP_{L2} is enabled on the ambient CPU:
      - · To configure the $\text{CDP}_{\text{L2-Data}}$ Capacity Bitmask for a COS, use `N=(COS<<1)`.
      - · To configure the $\text{CDP}_{\text{L2-Code}}$ Capacity Bitmask for a COS, use `N=(COS<<1)+1`.
  - For the L2 Capacity Bitmask, all (and only) contiguous combinations of 1-bits up to the highest capacity bit supported by the ambient CPU are valid.
- If **OP=7 (MBA Delay)**:
  - The descriptor `desc` uses the following encoding:

-	`MBA Delay`	`COS`
55 32	31 16	15 0

  - COS designates the CPU-local Class Of Service (COS) that shall be configured. Only COS below COS_{MB} of the ambient CPU are valid.
  - For the MBA Delay, only values up to the highest delay supported by the ambient CPU are valid.

If the hypercall completed successfully:

- If **OP=0 (S-State Transition)**:
  - The platform enters the specified ACPI sleep state or resets.
  - For shallow sleep states, the hypercall returns upon a wakeup event. The caller is responsible for invoking the necessary post-sleep ACPI methods and for transitioning platform devices back into the D0 working state.
  - For deep sleep states or platform reset, the hypercall does not return.
- If **OP=4 (QOS Configuration)**:
  - The ambient CPU uses and locks down the QOS configuration.
- If **OP=5 (CAT/CDP L3 Capacity Bitmask)**:
  - The ambient CPU uses the L3 Capacity Bitmask for the designated COS.

- If **OP=6 (CAT/CDP L2 Capacity Bitmask)**:
  - The ambient CPU uses the L2 Capacity Bitmask for the designated COS.
- If **OP=7 (MBA Delay)**:
  - The ambient CPU uses the MBA Delay for the designated COS.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_HYP

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

BAD_FTR

- The requested feature is not supported by the platform.

BAD_PAR

- If **OP=4**: The QOS configuration is invalid.
- If **OP=5/6/7**: The COS, CAT/CDP capacity bitmask or MBA delay is invalid.
- Otherwise: The requested operation (OP) is invalid.

ABORTED

- If **OP=0**: A concurrent power management transition prevailed.
- If **OP=4**: A QOS configuration has already been established for the ambient CPU.
- If **OP=5/6/7**: A QOS configuration has not yet been established for the ambient CPU.

5.5.2 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:

- { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `sm` } must refer to an SM Object Capability ($\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{SM}}}$) with permission ASSIGN.
- $\mathrm{CAP}_{\mathrm{OBJ}_{\mathrm{SM}}}$ must refer to an interrupt semaphore and thereby designates the interrupt.

If the hypercall completed successfully:

- The interrupt referred to by { $\mathrm{PD}_{\mathrm{CURRENT}}$, $\mathrm{SEL}_{\mathrm{OBJ}}$ `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.

Prior to the first invocation of `assign_int` for an interrupt, the state of that interrupt is as follows:

- the interrupt is masked.
- trigger, polarity and ownership are undefined.
- target CPU and authorized device are undefined.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CPU

- The specified CPU number was invalid.

BAD_CAP

- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} `sm` } did not refer to an SM Object Capability ($\text{CAP}_{\text{OBJ}_{\text{SM}}}$) or that capability had insufficient permissions.
- $\text{CAP}_{\text{OBJ}_{\text{SM}}}$ did not refer to an interrupt semaphore.

5.5.3 Assign Device

Parameters:

```
status = assign_dev (SEL_OBJ   pd,        // Protection Domain
                     SEL_MEM   smmu,      // SMMU Address (Page Number)
                     DAD       dad,       // Device Assignment Descriptor
                     TYPE_ACC  acc);      // Access Type
```

Flags:

0	0	0	0
3	2	1	0

Description:

Assigns a device to the specified Protection Domain (PD).

Prior to the hypercall:

- $\text{PD}_{\text{CURRENT}}$ must be the Root Protection Domain (PD_{ROOT}).
- { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} pd } must refer to a PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) with permission ASSIGN.
- { PD_{NOVA}, SEL_{MEM} smmu } must refer to the physical address of an SMMU/IOMMU.
- DAD designates the device and, if applicable, SMMU resources to use for managing that device.
- TYPE_{ACC} acc must refer to a DMA access type.

If the hypercall completed successfully:

- The device designated by DAD has been assigned to the Protection Domain (PD) referred to by { $\text{PD}_{\text{CURRENT}}$, SEL_{OBJ} pd }, such that DMA transactions of that device will be translated by the DMA page table corresponding to acc of that PD.
- DMA transactions of that device will be managed using the SMMU resources encoded in DAD. 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

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

BAD_PAR

- At least one of the parameters dad or acc was invalid.

6 Booting

6.1 Microhypervisor

6.1.1 ELF Image Loading

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

```
readelf -l nova.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 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 2 MiB, i.e. $\texttt{PhysAddr}_{\text{NEW}} = \texttt{PhysAddr}_{\text{ELF}} \pm \texttt{n} \times \texttt{2 MiB}$.
- 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).

6.1.2 Platform Resource Access

Possession of a PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) for PD_{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 (PD_{NOVA}):

Physical Memory

{ PD_{NOVA}, SEL_{MEM} $\texttt{0...PHYS}_{\texttt{NUM}}\texttt{-1}$ } refer to CAP_{MEM} for page frames in physical memory, where $\texttt{PHYS}_{\texttt{NUM}}$ is the number of page frames supported by the platform. Physical memory regions protected by the NOVA microhypervisor (Arm, x86) cannot be taken.

Interrupt Semaphores

{ PD_{NOVA}, SEL_{OBJ} $\texttt{1024...1024+INT}_{\texttt{PIN}}\texttt{-1}$ } refer to $\text{CAP}_{\text{OBJ}_{\text{SM}}}$ for interrupt semaphores, where $\texttt{INT}_{\texttt{PIN}}$ is the number of available pin-signaled interrupts, as conveyed by the HIP.

{ PD_{NOVA}, SEL_{OBJ} $\texttt{1024+INT}_{\texttt{PIN}}\texttt{...1024+INT}_{\texttt{PIN}}\texttt{+INT}_{\texttt{MSI}}\texttt{-1}$ } refer to $\text{CAP}_{\text{OBJ}_{\text{SM}}}$ for interrupt semaphores, where $\texttt{INT}_{\texttt{MSI}}$ is the number of available message-signaled interrupts, as conveyed by the HIP.

These capabilities can be used with the `ctrl_sm` and `assign_int` hypercalls.

Console Signaling Semaphore

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

6.2 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 is responsible for bootstrapping the other components of the user-mode framework by creating additional kernel objects, loading additional images, assigning resources, etc.

6.2.1 ELF Image Format

The ELF image of the Root Protection Domain (PD_{ROOT}) must be an executable (`ET_EXEC`) file that has been compiled for the respective architecture and

- linked such that `p_filesz = p_memsz`
- loaded such that `p_vaddr ≡ LOAD_ADDR`[*] `+ p_offset (mod PAGE_SIZE)`

holds for each loadable (`PT_LOAD`) program segment. These constraints ensure that the NOVA microhypervisor can map all program segments directly from physical into virtual memory without any additional memory allocation or copying. The following is an example:

```
readelf -l root.elf

Elf file type is EXEC (Executable file)
Entry point 0x10000120
There are 2 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz             Flags  Align
  LOAD           0x0000000000000000 0x0000000010000000 0x0000000010000000
                 0x0000000000000a75 0x0000000000000a75 R E    0x1000
  LOAD           0x0000000000001000 0x0000000010001000 0x0000000010001000
                 0x000000000000f004 0x000000000000f004 RW     0x1000
```

6.2.2 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.

6.2.2.1 Object Space

The object space contains the following initial capabilities:

- { PD_{ROOT}, SEL_{OBJ} $\mathtt{SEL}_{\mathtt{NUM}}\mathtt{-1}$ } refers to a PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) for PD_{NOVA}.
- { PD_{ROOT}, SEL_{OBJ} $\mathtt{SEL}_{\mathtt{NUM}}\mathtt{-2}$ } refers to a PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) for PD_{ROOT}.
- { PD_{ROOT}, SEL_{OBJ} $\mathtt{SEL}_{\mathtt{NUM}}\mathtt{-3}$ } refers to a EC Object Capability ($\text{CAP}_{\text{OBJ}_{\text{EC}}}$) for EC_{ROOT}.
- { PD_{ROOT}, SEL_{OBJ} $\mathtt{SEL}_{\mathtt{NUM}}\mathtt{-4}$ } refers to a SC Object Capability ($\text{CAP}_{\text{OBJ}_{\text{SC}}}$) for SC_{ROOT}.

All other { PD_{ROOT}, SEL_{OBJ} } refer to a Null Capability (CAP_0).

The value of $\mathtt{SEL}_{\mathtt{NUM}}$ is conveyed in the Hypervisor Information Page.

[*] This is the address in physical memory at which the bootloader has placed the ELF image.

6.2.2.2 Memory Space

ELF Program Segments

The microhypervisor maps the Root Protection Domain (PD_{ROOT}) into virtual memory according to the virtual addresses (`p_vaddr`), memory sizes (`p_memsz`) and page attributes (`p_flags`) of all loadable (`PT_LOAD`) program segments defined in the PD_{ROOT} ELF image.

Hypervisor Information Page

The microhypervisor maps the Hypervisor Information Page read-only into the memory space 4 KiB below the end of user-accessible virtual memory. The virtual address of the HIP is passed to EC_{ROOT} at the entry point (Arm, x86).

UTCB

The microhypervisor maps the User Thread Control Block of EC_{ROOT} into the memory space 4 KiB below the address of the Hypervisor Information Page.

All other { PD_{ROOT}, SEL_{MEM} } refer to a Null Capability (CAP_0).

6.3 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				+0x78
Features				+0x70
INT_{MSI}	INT_{PIN}	CPU_{BSP}	CPU_{NUM}	+0x68
$\text{SEL}_{\text{GST/NOVA}}$	$\text{SEL}_{\text{GST/ARCH}}$	$\text{SEL}_{\text{HST/NOVA}}$	$\text{SEL}_{\text{HST/ARCH}}$	+0x60
SEL_{NUM}				+0x58
STC 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
63 48	47 32	31 16	15 0	

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 [6] Root System Description Pointer (`0xffffffffffffffff` if not present).

UEFI Memory Map Address
Physical address of the UEFI [7] 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).

STC Frequency
Frequency of the System Time Counter (STC) 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.

CPU_{BSP}
The Bootstrap Processor (BSP) on which EC_{ROOT} and SC_{ROOT} have been created.

INT_{PIN}
Total number of pin-signaled interrupts that can be used via interrupt semaphores.

INT_{MSI}
Total number of message-signaled interrupts that can be used via interrupt semaphores.

Features
Supported platform features.

Architecture-Dependent
Architecture-dependent part. (Arm, x86)

Part IV

Application Binary Interface

7 ABI aarch64

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 entry point. 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 physical memory region occupied by the microhypervisor image must be clean to the PoC.
- 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.

7.1.1.1 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`) [8]
`X1`	Physical address of the Multiboot v2 information structure [8]
`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
`IP`	Physical address of the NOVA Protection Domain (PD_{NOVA}) ELF image entry point
`X0`	Multiboot v1 magic value (`0x2badb002`) [9]
`X1`	Physical address of the Multiboot v1 information structure [9]
`Other`	~

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

7.1.1.3 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 (FDT) for the hardware platform†
`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.

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 (EC$_{\text{ROOT}}$):

Register	Value / Description
IP	Virtual address of the Root Protection Domain (PD$_{\text{ROOT}}$) 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.

7.2 Protected Resources

The following resources are protected by the NOVA microhypervisor and are therefore inaccessible to user-mode applications.

7.2.1 Memory Space

Physical memory regions occupied by:

- NOVA microhypervisor – conveyed via HIP.
- GICD, GICR, GICC, GICH devices [10, 11] – conveyed via ACPI MADT or via FDT.
- SMMU devices [12, 13] – conveyed via ACPI IORT or via FDT.
- Firmware runtime services – conveyed via UEFI memory map.

7.3 Physical Memory

7.3.1 Memory Map

The Root Protection Domain (PD_{ROOT}) can obtain a list of available/reserved memory regions as follows:

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

7.4 Virtual Memory

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

7.4.1 Cacheability Attributes

Encoding	ATTR_{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 [3] for details on the architectural behavior.

7.4.2 Shareability Attributes

Encoding	ATTR_{SH}	Description
`0x0`	`NONE`	Not Shareable
`0x1`	-	*reserved*
`0x2`	`OUTER`	Outer Shareable
`0x3`	`INNER`	Inner Shareable

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

7.5 Class Of Service

Class Of Service (COS) is currently not supported.

7.6 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 the affected Execution Context's Event Selector Base (SEL_{EVT}).
- indexes into the Object Space (SPC_{OBJ}) of the affected EC's Protection Domain (PD).
- must refer to a PT Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PT}}}$) with permission `EVENT` that is bound to an EC on the same core as the affected EC, otherwise the affected EC is killed.

7.6.1 Architectural Events

Host Exceptions and Guest Intercepts

SEL_{OBJ}	Exception / Intercept
SEL_{EVT} + `0x00`	Unknown Reason
SEL_{EVT} + `0x01`	Trapped WFI or WFE
SEL_{EVT} + `0x02`	reserved
SEL_{EVT} + `0x03`	Trapped MCR or MRC
SEL_{EVT} + `0x04`	Trapped MCRR or MRRC
SEL_{EVT} + `0x05`	Trapped MCR or MRC
SEL_{EVT} + `0x06`	Trapped LDC or STC
SEL_{EVT} + `0x07`	SME, SVE, SIMD, FPU
SEL_{EVT} + `0x08`	Trapped VMRS Access
SEL_{EVT} + `0x09`	Trapped PAuth Instruction
SEL_{EVT} + `0x0a`	Trapped LD64B or ST64B
SEL_{EVT} + `0x0b`	reserved
SEL_{EVT} + `0x0c`	Trapped MRRC
SEL_{EVT} + `0x0d`	Branch Target Exception
SEL_{EVT} + `0x0e`	Illegal Execution State
SEL_{EVT} + `0x0f`	reserved
SEL_{EVT} + `0x10`	reserved
SEL_{EVT} + `0x11`	SVC (from AArch32 State)
SEL_{EVT} + `0x12`	HVC (from AArch32 State)
SEL_{EVT} + `0x13`	SMC (from AArch32 State)
SEL_{EVT} + `0x14`	reserved
SEL_{EVT} + `0x15`	SVC (from AArch64 State)*
SEL_{EVT} + `0x16`	HVC (from AArch64 State)
SEL_{EVT} + `0x17`	SMC (from AArch64 State)
SEL_{EVT} + `0x18`	Trapped MSR or MRS
SEL_{EVT} + `0x19`	Trapped SVE
SEL_{EVT} + `0x1a`	Trapped ERET
SEL_{EVT} + `0x1b`	TSTART Exception
SEL_{EVT} + `0x1c`	PAuth Instruction Failure
SEL_{EVT} + `0x1d`	Trapped SME
SEL_{EVT} + `0x1e`	Granule Protection Exception
SEL_{EVT} + `0x1f`	reserved
SEL_{EVT} + `0x20`	Instruction Abort (lower EL)
SEL_{EVT} + `0x21`	Instruction Abort (same EL)*
SEL_{EVT} + `0x22`	PC Alignment Fault
SEL_{EVT} + `0x23`	reserved
SEL_{EVT} + `0x24`	Data Abort (lower EL)
SEL_{EVT} + `0x25`	Data Abort (same EL)*
SEL_{EVT} + `0x26`	SP Alignment Fault
SEL_{EVT} + `0x27`	Memory Operation Exception
SEL_{EVT} + `0x28`	Trapped FPU (AArch32)
SEL_{EVT} + `0x29`	reserved
SEL_{EVT} + `0x2a`	reserved
SEL_{EVT} + `0x2b`	reserved
SEL_{EVT} + `0x2c`	Trapped FPU (AArch64)
SEL_{EVT} + `0x2d`	reserved
SEL_{EVT} + `0x2e`	reserved
SEL_{EVT} + `0x2f`	SError
SEL_{EVT} + `0x30`	Breakpoint (lower EL)
SEL_{EVT} + `0x31`	Breakpoint (same EL)*
SEL_{EVT} + `0x32`	Software Step (lower EL)
SEL_{EVT} + `0x33`	Software Step (same EL)*
SEL_{EVT} + `0x34`	Watchpoint (lower EL)
SEL_{EVT} + `0x35`	Watchpoint (same EL)*
SEL_{EVT} + `0x36`	reserved
SEL_{EVT} + `0x37`	reserved
SEL_{EVT} + `0x38`	BKPT (AArch32)
SEL_{EVT} + `0x39`	reserved
SEL_{EVT} + `0x3a`	Vector Catch (AArch32)
SEL_{EVT} + `0x3b`	reserved
SEL_{EVT} + `0x3c`	BRK (AArch64)
SEL_{EVT} + `0x3d`	reserved
SEL_{EVT} + `0x3e`	reserved
SEL_{EVT} + `0x3f`	reserved

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

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

7.6.2 Microhypervisor Events

$\mathbf{SEL_{OBJ}}$	**Event**
$\mathrm{SEL_{EVT}} + \mathtt{SEL_{ARCH}} + \mathtt{0x0}$	Startup
$\mathrm{SEL_{EVT}} + \mathtt{SEL_{ARCH}} + \mathtt{0x1}$	Recall
$\mathrm{SEL_{EVT}} + \mathtt{SEL_{ARCH}} + \mathtt{0x2}$	Virtual Timer

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

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

7.7 Architecture-Dependent Structures

7.7.1 Hypervisor Information Page

63 32	31 16	15 0	+Length
~	$\mathtt{CTX}_{\mathtt{NUM}}$	$\mathtt{SMG}_{\mathtt{NUM}}$	Arch+0x00
63 32	31 16	15 0	

SMG$_{\text{NUM}}$

Total number of Stream Mapping Groups (SMGs).

CTX$_{\text{NUM}}$

Total number of Translation Contexts (CTXs).

7.7.2 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	
-	HSTR	IFSR	DACR	+0x120	A32
SPSR_und	SPSR_irq	SPSR_fiq	SPSR_abt	+0x110	
TPIDRRO_EL0		TPIDR_EL0		+0x100	EL0
SP_EL0		X30 (LR_fiq)		+0x0f0	
X29 (SP_fiq)		X28 (R12_fiq)		+0x0e0	
X27 (R11_fiq)		X26 (R10_fiq)		+0x0d0	
X25 (R9_fiq)		X24 (R8_fiq)		+0x0c0	
X23 (SP_und)		X22 (LR_und)		+0x0b0	
X21 (SP_abt)		X20 (LR_abt)		+0x0a0	
X19 (SP_svc)		X18 (LR_svc)		+0x090	
X17 (SP_irq)		X16 (LR_irq)		+0x080	
X15 (SP_hyp)		X14 (LR_usr)		+0x070	
X13 (SP_usr)		X12 (R12_usr)		+0x060	
X11 (R11_usr)		X10 (R10_usr)		+0x050	
X9 (R9_usr)		X8 (R8_usr)		+0x040	
X7 (R7)		X6 (R6)		+0x030	
X5 (R5)		X4 (R4)		+0x020	
X3 (R3)		X2 (R2)		+0x010	
X1 (R1)		X0 (R0)		+0x000	

48 32 16 0 48 32 16 0

7.7.3 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 4.4.2, 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_DIH	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 (7.7.2) 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 Thread	Kills the vCPU
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_DIH	rw	-	DACR, IFSR, HSTR
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 the condition flags are writable.

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

7.7.4 Device Assignment Descriptor

The Device Assignment Descriptor (DAD) describes a device to be assigned to a Protection Domain (PD).

On Arm, it also specifies the SMMU resources that should be used to manage DMA transactions of that device.

-	CTX	SMG	SID Mask	SID
63 48	47 40	39 32	31 16	15 0

The fields are defined as follows:

SID

Designates the device via its Stream Identifier (SID).

SID Mask

Specifies which bits of that SID should be matched (0) or ignored (1) by the Stream Mapping Group.

SMG

Specifies the Stream Mapping Group (SMG) to use for that SID – must be $< \text{SMG}_{\text{NUM}}$.

CTX

Specifies the Translation Context (CTX) to use for that SMG – must be $< \text{CTX}_{\text{NUM}}$.

System software must ensure an unambiguous assignment of Stream Identifiers to Stream Mapping Groups, i.e. it must configure the SID/Mask fields across all Stream Mapping Groups such that no SID multi-matches can occur.

7.8 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 CPU registers and then executes the `svc #0` instruction [3].

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

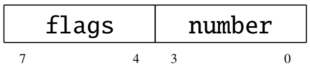


Figure 7.1: Hypercall Identifier

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

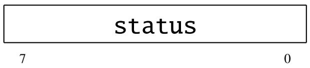


Figure 7.2: Status Code

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

IPC Call

pt$_{[63-8]}$ hypercall$_{[7-0]}$	X0	$\xrightarrow{\text{ipc\_call}}$	X0	status$_{[7-0]}$
mtd$_{[31-0]}$	X1		X1	mtd$_{[31-0]}$
-	IP	svc #0	IP	IP+4

IPC Reply

hypercall$_{[7-0]}$	X0	$\xrightarrow{\text{ipc\_reply}}$	X0	pid
mtd$_{[31-0]}$	X1		X1	mtd$_{[31-0]}$
-	IP	svc #0	IP	Portal IP

Create Protection Domain

sel$_{[63-8]}$ hypercall$_{[7-0]}$	X0	$\xrightarrow{\text{create\_pd}}$	X0	status$_{[7-0]}$
own	X1		X1	≡
-	IP	svc #0	IP	IP+4

Create Execution Context

sel$_{[63-8]}$ hypercall$_{[7-0]}$	X0	$\xrightarrow{\text{create\_ec}}$	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 →	X0	status$_{[7-0]}$
own	X1		X1	≡
ec	X2		X2	≡
scd	X3		X3	≡
-	IP	svc #0	IP	IP+4

Create Portal

sel$_{[63-8]}$ hypercall$_{[7-0]}$	X0	create_pt →	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 →	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 →	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$_{[6-2]}$ acc$_{[1-0]}$	X3		X3	≡
-	IP	svc #0	IP	IP+4

Control Execution Context

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

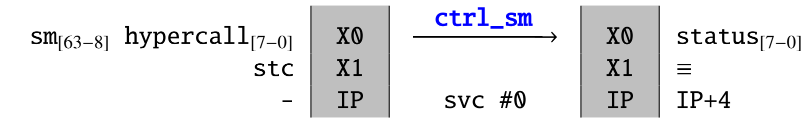
Control Scheduling Context

sc$_{[63-8]}$ hypercall$_{[7-0]}$	X0	ctrl_sc →	X0	status$_{[7-0]}$
-	X1		X1	stc
-	IP	svc #0	IP	IP+4

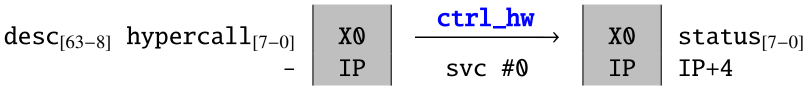
Control Portal

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

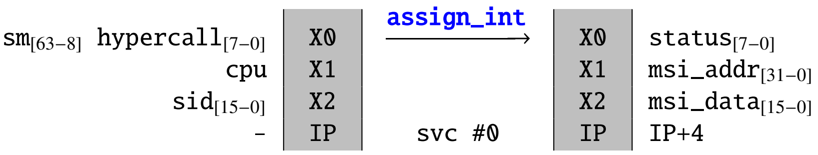
Control Semaphore



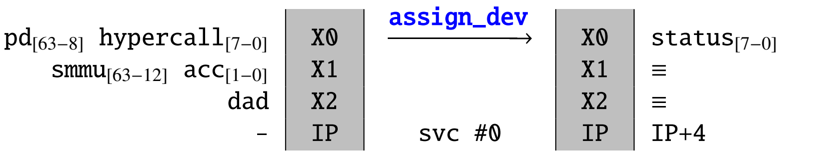
Control Hardware



Assign Interrupt



Assign Device



7.9 Supplementary Functionality

This section describes functions that do **not** conform to the calling convention for hypercalls. Because these functions cannot perform capability-based access control, their invocation is restricted to the Root Protection Domain (PD_{ROOT}). Invocation of these functions from any other Protection Domain generates an exception.

Secure Monitor Call

identifier$_{[31-0]}$	X0	$\xrightarrow{\texttt{proxy\_smc}}$	X0	~
-	X1		X1	~
-	X2		X2	~
-	X3		X3	~
-	X4		X4	~
-	X5		X5	~
-	X6		X6	~
-	X7		X7	~
-	X8		X8	~
-	X9		X9	~
-	X10		X10	~
-	X11		X11	~
-	X12		X12	~
-	X13		X13	~
-	X14		X14	~
-	X15		X15	~
-	X16		X16	~
-	X17		X17	~
-	IP	svc #1	IP	IP+4

This call is proxy-filtered by the microhypervisor. If the combination of invoked service (identifier$_{[29-24]}$) and function (identifier$_{[15-0]}$) is listed in the table below, then the microhypervisor issues the corresponding SMC to platform firmware on behalf of the caller. Otherwise, this function generates an exception. Register allocation conforms to the Arm SMCCC [14].

Service	Description	Function	Description
0x2	SIP Service Calls	0x0000-0xffff	All functions
0x4	Standard Secure Service Calls	0x0050-0x005f	TRNG functions [15]
		0x0130-0x013f	PCI functions [16]
0x30-0x31	Trusted Application Calls	0x0000-0xffff	All functions
0x32-0x3f	Trusted OS Calls	0x0000-0xffff	All functions

8 ABI x86-64

8.1 Boot State

8.1.1 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 entry point. Furthermore, the following preconditions must be satisfied:

- The CPU state must conform to a machine state defined in the Multiboot Specification v2 [8] or v1 [9].
- 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.

8.1.1.1 Multiboot v2 Launch

Only this launch type supports 64-bit UEFI platforms.

Register	Value / Description
`EIP`	Physical address of the NOVA Protection Domain (PD_{NOVA}) ELF image entry point
`EAX`	Multiboot v2 magic value (`0x36d76289`) [8]
`EBX`	Physical address of the Multiboot v2 information structure [8]
`Other`	~

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

8.1.1.2 Multiboot v1 Launch

Register	Value / Description
`EIP`	Physical address of the NOVA Protection Domain (PD_{NOVA}) ELF image entry point
`EAX`	Multiboot v1 magic value (`0x2badb002`) [9]
`EBX`	Physical address of the Multiboot v1 information structure [9]
`Other`	~

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

8.1.2 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
`RIP`	Virtual address of the Root Protection Domain (PD_{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.

8.2 Protected Resources

The following resources are protected by the NOVA microhypervisor and are therefore inaccessible to user-mode applications.

8.2.1 Memory Space

Physical memory regions occupied by:

- NOVA microhypervisor – conveyed via HIP.
- LAPIC, IOAPIC devices – conveyed via ACPI MADT.
- IOMMU devices [17, 18] – conveyed via ACPI DMAR or IVRS.
- Firmware runtime services – conveyed via UEFI memory map.

8.2.2 I/O Port Space

- ACPI fixed registers `PM1a_CNT`, `PM1b_CNT`, `PM2_CNT` – conveyed via ACPI FADT.
- `SMI_CMD` port – conveyed via ACPI FADT.

8.3 Physical Memory

8.3.1 Memory Map

The Root Protection Domain (PD_{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 [7].
- On platforms using Multiboot v2, by parsing the Multiboot v2 memory map [8].
- On platforms using Multiboot v1, by parsing the Multiboot v1 memory map [9].

8.4 Virtual Memory

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

8.4.1 Cacheability Attributes

Encoding	ATTR_{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 [4, 5] for details on the architectural behavior.

8.4.2 Shareability Attributes

Encoding	ATTR_{SH}	Description
`0x0`	`UNUSED`	Always use this value

8.5 Class Of Service

Class Of Service (COS) support is indicated by CPUID leaf `0x7`, sub-leaf `0x0`: $\texttt{EBX}_{[15]}$.

The Root Protection Domain (PD_{ROOT}) must perform the following steps on each CPU to configure QOS settings:

1. Invoke `ctrl_hw` to establish a valid QOS configuration:
   - CDP_{L3} support is indicated by CPUID leaf `0x10`, sub-leaf `0x1`: $\texttt{ECX}_{[2]}$
   - CDP_{L2} support is indicated by CPUID leaf `0x10`, sub-leaf `0x2`: $\texttt{ECX}_{[2]}$
2. Determine COS_{NUM} as the maximum of the following:
   - COS_{L3} from CPUID leaf `0x10`, sub-leaf `0x1`: $\texttt{(1+EDX}_{[15-0]}\texttt{)>>X}$, where
     - X=0 if CDP_{L3} is disabled.
     - X=1 if CDP_{L3} is enabled.
   - COS_{L2} from CPUID leaf `0x10`, sub-leaf `0x2`: $\texttt{(1+EDX}_{[15-0]}\texttt{)>>X}$, where
     - X=0 if CDP_{L2} is disabled.
     - X=1 if CDP_{L2} is enabled.
   - COS_{MB} from CPUID leaf `0x10`, sub-leaf `0x3`: $\texttt{(1+EDX}_{[15-0]}\texttt{)}$
3. Invoke `ctrl_hw` to configure the following:
   - For each COS_{L3}: CAT/CDP L3 Capacity Bitmask(s)
   - For each COS_{L2}: CAT/CDP L2 Capacity Bitmask(s)
   - For each COS_{MB}: MBA Delay

8.6 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 the affected Execution Context's Event Selector Base (SEL_{EVT}).
- indexes into the Object Space (SPC_{OBJ}) of the affected EC's Protection Domain (PD).
- must refer to a PT Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PT}}}$) with permission `EVENT` that is bound to an EC on the same core as the affected EC, otherwise the affected EC is killed.

8.6.1 Architectural Events

Host Exceptions

SEL_{OBJ}	Exception
SEL_{EVT} + `0x00`	`#DE`
SEL_{EVT} + `0x01`	`#DB`
SEL_{EVT} + `0x02`	reserved
SEL_{EVT} + `0x03`	`#BP`
SEL_{EVT} + `0x04`	`#OF`
SEL_{EVT} + `0x05`	`#BR`
SEL_{EVT} + `0x06`	`#UD`
SEL_{EVT} + `0x07`	`#NM`[*]
SEL_{EVT} + `0x08`	`#DF`[*]
SEL_{EVT} + `0x09`	reserved
SEL_{EVT} + `0x0a`	`#TS`[*]
SEL_{EVT} + `0x0b`	`#NP`
SEL_{EVT} + `0x0c`	`#SS`
SEL_{EVT} + `0x0d`	`#GP`
SEL_{EVT} + `0x0e`	`#PF`
SEL_{EVT} + `0x0f`	reserved
SEL_{EVT} + `0x10`	`#MF`
SEL_{EVT} + `0x11`	`#AC`
SEL_{EVT} + `0x12`	`#MC`[*]
SEL_{EVT} + `0x13`	`#XM`
SEL_{EVT} + `0x14`	`#VE`
SEL_{EVT} + `0x15`	`#CP`
SEL_{EVT} + `0x16`	reserved
SEL_{EVT} + `0x17`	reserved
SEL_{EVT} + `0x18`	reserved
SEL_{EVT} + `0x19`	reserved
SEL_{EVT} + `0x1a`	reserved
SEL_{EVT} + `0x1b`	reserved
SEL_{EVT} + `0x1c`	reserved
SEL_{EVT} + `0x1d`	reserved
SEL_{EVT} + `0x1e`	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$_{\text{OBJ}}$	Intercept
SEL$_{\text{EVT}}$ + 0x00	Exception or NMI*
SEL$_{\text{EVT}}$ + 0x01	External Interrupt*
SEL$_{\text{EVT}}$ + 0x02	Triple Fault†
SEL$_{\text{EVT}}$ + 0x03	INIT†
SEL$_{\text{EVT}}$ + 0x04	SIPI†
SEL$_{\text{EVT}}$ + 0x05	I/O SMI
SEL$_{\text{EVT}}$ + 0x06	Other SMI
SEL$_{\text{EVT}}$ + 0x07	Interrupt Window
SEL$_{\text{EVT}}$ + 0x08	NMI Window
SEL$_{\text{EVT}}$ + 0x09	Task Switch†
SEL$_{\text{EVT}}$ + 0x0a	CPUID†
SEL$_{\text{EVT}}$ + 0x0b	GETSEC†
SEL$_{\text{EVT}}$ + 0x0c	HLT†
SEL$_{\text{EVT}}$ + 0x0d	INVD†
SEL$_{\text{EVT}}$ + 0x0e	INVLPG
SEL$_{\text{EVT}}$ + 0x0f	RDPMC
SEL$_{\text{EVT}}$ + 0x10	RDTSC
SEL$_{\text{EVT}}$ + 0x11	RSM
SEL$_{\text{EVT}}$ + 0x12	VMCALL
SEL$_{\text{EVT}}$ + 0x13	VMCLEAR
SEL$_{\text{EVT}}$ + 0x14	VMLAUNCH
SEL$_{\text{EVT}}$ + 0x15	VMPTRLD
SEL$_{\text{EVT}}$ + 0x16	VMPTRST
SEL$_{\text{EVT}}$ + 0x17	VMREAD
SEL$_{\text{EVT}}$ + 0x18	VMRESUME
SEL$_{\text{EVT}}$ + 0x19	VMWRITE
SEL$_{\text{EVT}}$ + 0x1a	VMXOFF
SEL$_{\text{EVT}}$ + 0x1b	VMXON
SEL$_{\text{EVT}}$ + 0x1c	CR Access*
SEL$_{\text{EVT}}$ + 0x1d	DR Access
SEL$_{\text{EVT}}$ + 0x1e	I/O Access†
SEL$_{\text{EVT}}$ + 0x1f	RDMSR†
SEL$_{\text{EVT}}$ + 0x20	WRMSR†
SEL$_{\text{EVT}}$ + 0x21	VM Entry Failure (State)†
SEL$_{\text{EVT}}$ + 0x22	VM Entry Failure (MSR)
SEL$_{\text{EVT}}$ + 0x23	reserved
SEL$_{\text{EVT}}$ + 0x24	MWAIT
SEL$_{\text{EVT}}$ + 0x25	MTF
SEL$_{\text{EVT}}$ + 0x26	reserved
SEL$_{\text{EVT}}$ + 0x27	MONITOR
SEL$_{\text{EVT}}$ + 0x28	PAUSE
SEL$_{\text{EVT}}$ + 0x29	VM Entry Failure (MCE)
SEL$_{\text{EVT}}$ + 0x2a	reserved
SEL$_{\text{EVT}}$ + 0x2b	TPR Below Threshold
SEL$_{\text{EVT}}$ + 0x2c	APIC Access
SEL$_{\text{EVT}}$ + 0x2d	Virtualized EOI
SEL$_{\text{EVT}}$ + 0x2e	GDTR/IDTR Access
SEL$_{\text{EVT}}$ + 0x2f	LDTR/TR Access
SEL$_{\text{EVT}}$ + 0x30	EPT Violation†
SEL$_{\text{EVT}}$ + 0x31	EPT Misconfiguration
SEL$_{\text{EVT}}$ + 0x32	INVEPT
SEL$_{\text{EVT}}$ + 0x33	RDTSCP
SEL$_{\text{EVT}}$ + 0x34	Preemption Timer
SEL$_{\text{EVT}}$ + 0x35	INVVPID
SEL$_{\text{EVT}}$ + 0x36	WBINVD, WBNOINVD
SEL$_{\text{EVT}}$ + 0x37	XSETBV
SEL$_{\text{EVT}}$ + 0x38	APIC Write
SEL$_{\text{EVT}}$ + 0x39	RDRAND
SEL$_{\text{EVT}}$ + 0x3a	INVPCID
SEL$_{\text{EVT}}$ + 0x3b	VMFUNC
SEL$_{\text{EVT}}$ + 0x3c	ENCLS
SEL$_{\text{EVT}}$ + 0x3d	RDSEED
SEL$_{\text{EVT}}$ + 0x3e	PML Log Full
SEL$_{\text{EVT}}$ + 0x3f	XSAVES
SEL$_{\text{EVT}}$ + 0x40	XRSTORS
SEL$_{\text{EVT}}$ + 0x41	reserved
SEL$_{\text{EVT}}$ + 0x42	SPP Miss / Misconfiguration
SEL$_{\text{EVT}}$ + 0x43	UMWAIT
SEL$_{\text{EVT}}$ + 0x44	TPAUSE
SEL$_{\text{EVT}}$ + 0x45	LOADIWKEY
SEL$_{\text{EVT}}$ + 0x46	reserved
SEL$_{\text{EVT}}$ + 0x47	reserved
SEL$_{\text{EVT}}$ + 0x48	ENQCMD PASID Failure
SEL$_{\text{EVT}}$ + 0x49	ENQCMDS PASID Failure
SEL$_{\text{EVT}}$ + 0x4a	Bus Lock
SEL$_{\text{EVT}}$ + 0x4b	Notify Window
SEL$_{\text{EVT}}$ + 0x4c	SEAMCALL
SEL$_{\text{EVT}}$ + 0x4d	TDCALL
SEL$_{\text{EVT}}$ + 0x4e	reserved
SEL$_{\text{EVT}}$ + 0x4f	reserved

Please refer to [4] for more details on each of these events.

8.6.2 Microhypervisor Events

SEL$_{\text{OBJ}}$	Event
SEL$_{\text{EVT}}$ + SEL$_{\text{ARCH}}$ + 0x0	Startup
SEL$_{\text{EVT}}$ + SEL$_{\text{ARCH}}$ + 0x1	Recall

The value of SEL$_{\text{ARCH}}$ depends on the origin of the event:

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

8.7 Architecture-Dependent Structures

8.7.1 Hypervisor Information Page

The architecture-dependent HIP structure is empty.

8.7.2 User Thread Control Block

-				IA32_KERNEL_GS_BASE				+0x240
IA32_FMASK				IA32_LSTAR				+0x230
IA32_STAR				IA32_EFER				+0x220
IA32_PAT				IA32_SYSENTER_EIP				+0x210
IA32_SYSENTER_ESP				IA32_SYSENTER_CS				+0x200
IA32_XSS				XCR0				+0x1f0
DR7				CR8				+0x1e0
CR4				CR3				+0x1d0
CR2				CR0				+0x1c0
PDPTE3				PDPTE2				+0x1b0
PDPTE1				PDPTE0				+0x1a0
Base IDTR				Limit IDTR		-		+0x190
Base GDTR				Limit GDTR		-		+0x180
Base LDTR				Limit LDTR		AR LDTR[*]	SEL LDTR	+0x170
Base TR				Limit TR		AR TR[*]	SEL TR	+0x160
Base GS				Limit GS		AR GS[*]	SEL GS	+0x150
Base FS				Limit FS		AR FS[*]	SEL FS	+0x140
Base ES				Limit ES		AR ES[*]	SEL ES	+0x130
Base DS				Limit DS		AR DS[*]	SEL DS	+0x120
Base SS				Limit SS		AR SS[*]	SEL SS	+0x110
Base CS				Limit CS		AR CS[*]	SEL CS	+0x100
IDT Vectoring Error		IDT Vectoring Info		Interruption Error		Interruption Info[†]		+0x0f0
TPR Threshold		PF Error Match		PF Error Mask		EXC Intercepts		+0x0e0
CR4 Intercepts				CR0 Intercepts				+0x0d0
3rd Exec Controls				2nd Exec Controls		1st Exec Controls		+0x0c0
-				3rd Qualification				+0x0b0
2nd Qualification				1st Qualification				+0x0a0
Activity		Interruptibility		Instruction Info		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 8.7.2.1 for encoding details.
[†] See Section 8.7.2.2 for encoding details.

8.7.2.1 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

8.7.2.2 Encoding: Interruption 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 Interruption Error field 1 = Deliver the error code from the UTCB Interruption 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

8.7.3 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 4.4.2, has the following layout:

FPU	TLB	-	KERNEL_GS	SYSCALL	EFER	PAT	SYSENTER	XSAVE	DR	CR	PDPTE	IDTR	GDTR	LDTR	TR	FS/GS	DS/ES	CS/SS	INJ	TPR	CTRL	QUAL	STA	RIP	RFLAGS	GPR$_{8-15}$	GPR$_{0-7}$	POISON
31	30		25	24	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 (8.7.2) 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 Thread	Kills the vCPU
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, Instruction Info
STA	rw	-	Interruptibility State, Activity State
QUAL	r	Qualifications[†]	Qualifications[‡]
CTRL	w	-	Execution Controls, CR Intercepts, EXC Intercepts PF Error Mask/Match
TPR	w	-	TPR Threshold
INJ	rw r	-	Interruption Info, Interruption Error IDT Vectoring Info, IDT Vectoring 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
XSAVE	rw	-	XCR0, IA32_XSS
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
TLB	w	-	Invalidates the TLB for the vCPU

[*] Only the status and control flags are writable.

[†] Qualification fields contain exception error code (1st), page-fault linear address (2nd).

[‡] Qualification fields contain exit qualification (1st), guest-linear address (2nd), guest-physical address (3rd).

8.7.4 Device Assignment Descriptor

The Device Assignment Descriptor (DAD) describes a device to be assigned to a Protection Domain (PD). On x86, IOMMU resources need not be specified.

-	B	D	F
63 16	15 8	7 3	2 0

The fields are defined as follows:

B, D, F:
Designates the device via its Bus/Device/Function (BDF) source identifier.

8.8 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 CPU registers and then executes the `syscall` instruction [4, 5].

The hypercall identifier consists of the hypercall number and hypercall-specific flags, as illustrated in Figure 8.1.

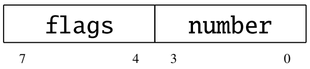


Figure 8.1: Hypercall Identifier

The status code returned from a hypercall has the format shown in Figure 8.2.

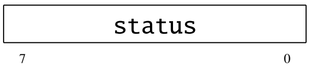


Figure 8.2: Status Code

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

IPC Call

pt$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	ipc_call →	RDI	status$_{[7-0]}$
mtd$_{[31-0]}$	RSI		RSI	mtd$_{[31-0]}$
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

IPC Reply

hypercall$_{[7-0]}$	RDI	ipc_reply →	RDI	pid
mtd$_{[31-0]}$	RSI		RSI	mtd$_{[31-0]}$
-	RCX		RCX	Portal IP
-	R11		R11	0x202
-	RIP	syscall	RIP	Portal IP

Create Protection Domain

sel$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	create_pd →	RDI	status$_{[7-0]}$
own	RSI		RSI	≡
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Create Execution Context

sel$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	create_ec →	RDI	status$_{[7-0]}$
own	RSI		RSI	≡
utcb$_{[63-12]}$ cpu$_{[11-0]}$	RDX		RDX	≡
sp	RAX		RAX	≡
evt	R8		R8	≡
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Create Scheduling Context

sel$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	create_sc →	RDI	status$_{[7-0]}$
own	RSI		RSI	≡
ec	RDX		RDX	≡
scd	RAX		RAX	≡
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Create Portal

sel$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	create_pt →	RDI	status$_{[7-0]}$
own	RSI		RSI	≡
ec	RDX		RDX	≡
ip	RAX		RAX	≡
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Create Semaphore

sel$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	create_sm →	RDI	status$_{[7-0]}$
own	RSI		RSI	≡
cnt	RDX		RDX	≡
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Control Protection Domain

spd$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	ctrl_pd →	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$_{[6-2]}$ acc$_{[1-0]}$	RAX		RAX	≡
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Control Execution Context

ec$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	ctrl_ec →	RDI	status$_{[7-0]}$
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Control Scheduling Context

sc$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	ctrl_sc →	RDI	status$_{[7-0]}$
-	RSI		RSI	stc
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Control Portal

pt$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	ctrl_pt →	RDI	status$_{[7-0]}$
pid	RSI		RSI	≡
mtd$_{[31-0]}$	RDX		RDX	≡
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Control Semaphore

sm$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	ctrl_sm →	RDI	status$_{[7-0]}$
stc	RSI		RSI	≡
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Control Hardware

desc$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	ctrl_hw →	RDI	status$_{[7-0]}$
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Assign Interrupt

sm$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	assign_int →	RDI	status$_{[7-0]}$
cpu	RSI		RSI	msi_addr$_{[31-0]}$
bdf$_{[15-0]}$	RDX		RDX	msi_data$_{[15-0]}$
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Assign Device

pd$_{[63-8]}$ hypercall$_{[7-0]}$	RDI	$\xrightarrow{\texttt{assign\_dev}}$	RDI	status$_{[7-0]}$
smmu$_{[63-12]}$ acc$_{[1-0]}$	RSI		RSI	≡
dad	RDX		RDX	≡
-	RCX		RCX	RIP+2
-	R11		R11	0x202
-	RIP	syscall	RIP	RIP+2

Part V

Appendix

A Acronyms

ACPI	Advanced Configuration and Power Interface [6]
ATTR$_{\text{CA}}$	Cacheability Attribute [Arm, x86]
ATTR$_{\text{SH}}$	Shareability Attribute [Arm, x86]
BDF	Bus/Device/Function [19, 20]
BSP	Bootstrap Processor
CAP	Capability
CAP$_{0}$	Null Capability
CAP$_{\text{MEM}}$	Memory Capability
CAP$_{\text{MSR}}$	MSR Capability
CAP$_{\text{OBJ}}$	Object Capability
CAP$_{\text{OBJ}_{\text{PD}}}$	PD Object Capability
CAP$_{\text{OBJ}_{\text{EC}}}$	EC Object Capability
CAP$_{\text{OBJ}_{\text{SC}}}$	SC Object Capability
CAP$_{\text{OBJ}_{\text{PT}}}$	PT Object Capability
CAP$_{\text{OBJ}_{\text{SM}}}$	SM Object Capability
CAP$_{\text{PIO}}$	I/O Port Capability
CAT	Cache Allocation Technology [4]
CDP	Code and Data Prioritization [4]
COS	Class Of Service [Arm, x86] [4]
CPU	Central Processing Unit [3, 4, 5]
CTX	Translation Context [12, 13]
DAD	Device Assignment Descriptor [Arm, x86]
DMA	Direct Memory Access
EC	Execution Context
EC$_{\text{CURRENT}}$	Current Execution Context
EC$_{\text{ROOT}}$	Root Execution Context
ELF	Executable and Linkable Format [21]
FDT	Flattened Device Tree [22]
FPU	Floating Point Unit [3, 4, 5]
GIC	Generic Interrupt Controller [10, 11]
GICC	GIC CPU Interface
GICD	GIC Distributor
GICH	GIC HYP Interface
GICR	GIC Redistributor
HIP	Hypervisor Information Page [Arm, x86]
IOAPIC	I/O Advanced Programmable Interrupt Controller
IOMMU	I/O Memory Management Unit[17, 18]

IP	Instruction Pointer
IPC	Inter-Process Communication
LAPIC	Local Advanced Programmable Interrupt Controller
MBA	Memory Bandwidth Allocation [4]
MMU	Memory Management Unit [3, 4, 5]
MSI	Message-Signaled Interrupt [19, 20]
MSR	Model-Specific Register [4, 5]
MTD	Message Transfer Descriptor [Arm, x86]
NOVA	NOVA **OS** **V**irtualization **A**rchitecture [2]
PCI	Peripheral Component Interconnect [19, 20]
PD	Protection Domain
$\text{PD}_{\text{CURRENT}}$	Current Protection Domain
PD_{NOVA}	NOVA Protection Domain
PD_{ROOT}	Root Protection Domain
PID	Portal Identifier
PT	Portal
QOS	Quality Of Service
SC	Scheduling Context
$\text{SC}_{\text{CURRENT}}$	Current Scheduling Context
SC_{ROOT}	Root Scheduling Context
SCD	Scheduling Context Descriptor
SEL	Capability Selector
SEL_{EVT}	Event Selector Base [Arm, x86]
SEL_{MEM}	Memory Capability Selector
SEL_{MSR}	MSR Capability Selector
SEL_{OBJ}	Object Capability Selector
SEL_{PIO}	I/O Port Capability Selector
SID	Stream Identifier [12, 13]
SM	Semaphore
SMG	Stream Mapping Group [12, 13]
SMMU	System Memory Management Unit [12, 13]
SP	Stack Pointer
SPC_{MEM}	Memory Space
SPC_{MSR}	MSR Space
SPC_{OBJ}	Object Space
SPC_{PIO}	I/O Port Space
STC	System Time Counter
TYPE_{SPC}	Space Type
TYPE_{ACC}	Access Type
UART	Universal Asynchronous Receiver Transmitter
UEFI	Unified Extensible Firmware Interface [7]
UTCB	User Thread Control Block [Arm, x86]

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] *RFC 2119*. Internet Engineering Task Force (IETF), 1997. URL https://tools.ietf.org/html/rfc2119. iv

[2] 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, 73

[3] *Arm Architecture Reference Manual ARMv8, for ARMv8-A Architecture Profile*. Arm Limited, 2022. URL https://developer.arm.com/documentation/ddi0487/. Document Number: DDI0487. 7, 46, 48, 54, 72, 73

[4] *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, 2021. URL https://software.intel.com/en-us/articles/intel-sdm. Document Number: 325462. 7, 59, 62, 67, 72, 73

[5] *AMD64 Architecture Programmer's Manual: Volumes 1–5*. Advanced Micro Devices, Inc., 2021. URL https://developer.amd.com/resources/developer-guides-manuals. Document Number: 40332. 7, 59, 67, 72, 73

[6] *Advanced Configuration and Power Interface (ACPI) Specification*. UEFI Forum, Inc., 2021. URL https://uefi.org/specifications. Version 6.4. 41, 72

[7] *Unified Extensible Firmware Interface (UEFI) Specification*. UEFI Forum, Inc., 2021. URL https://uefi.org/specifications. Version 2.9. 41, 59, 73

[8] 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. 44, 58, 59

[9] 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. 44, 58, 59

[10] *Arm Generic Interrupt Controller Architecture Specification Version 2*. Arm Limited, 2013. URL https://developer.arm.com/documentation/ihi0048/. Document Number: IHI0048. 46, 72

[11] *Arm Generic Interrupt Controller Architecture Specification Version 3 and Version 4*. Arm Limited, 2022. URL https://developer.arm.com/documentation/ihi0069/. Document Number: IHI0069. 46, 72

[12] *Arm System Memory Management Unit Architecture Specification Version 2*. Arm Limited, 2016. URL https://developer.arm.com/documentation/ihi0062/. Document Number: IHI0062. 46, 72, 73

[13] *Arm System Memory Management Unit Architecture Specification Version 3*. Arm Limited, 2021. URL https://developer.arm.com/documentation/ihi0070/. Document Number: IHI0070. 46, 72, 73

[14] *Arm SMC Calling Convention*. Arm Limited, 2022. URL https://developer.arm.com/documentation/den0028/. Document Number: DEN0028. 57

[15] *Arm True Random Number Generator Firmware Interface*. Arm Limited, 2022. URL https://developer.arm.com/documentation/den0098/. Document Number: DEN0098. 57

[16] *Arm PCI Configuration Space Access Firmware Interface*. Arm Limited, 2022. URL https://developer.arm.com/documentation/den0115/. Document Number: DEN0115. 57

[17] *Intel Virtualization Technology for Directed I/O Architecture Specification*. Intel Corporation, 2021. URL https://www.intel.com/content/www/us/en/develop/download/intel-virtualization-technology-for-directed-io-architecture-specification.html. Document Number: D51397. 59, 72

[18] *AMD I/O Virtualization Technology (IOMMU) Specification*. Advanced Micro Devices, Inc., 2021. URL https://www.amd.com/en/support/tech-docs/amd-io-virtualization-technology-iommu-specification. Document Number: 48882. 59, 72

[19] *PCI Local Bus Specification*. PCI-SIG, 2004. URL https://pcisig.com/specifications. Revision 3.0. 72, 73

[20] *PCI Express Base Specification*. PCI-SIG, 2019. URL https://pcisig.com/specifications. Revision 5.0. 72, 73

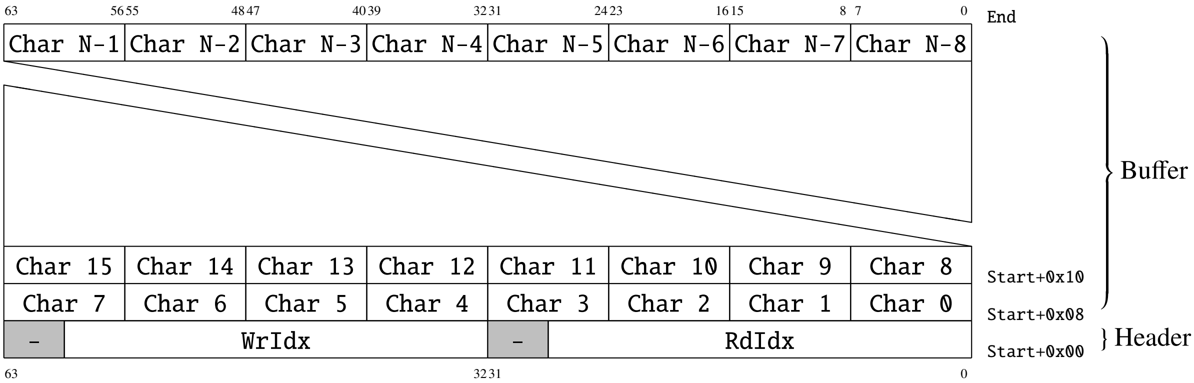
[21] *Executable and Linking Format (ELF) Specification*. TIS Committee, 1995. URL https://refspecs.linuxbase.org/elf/elf.pdf. Version 1.2. 72

[22] *Devicetree Specification*. Linaro Limited, 2020. URL https://www.devicetree.org/specifications. Version 0.3. 72

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 6.1.2) 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:

- RdIdx 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.
- WrIdx 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 RdIdx is equal to WrIdx.
- Otherwise WrIdx is ahead of RdIdx, 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 RdIdx, 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 WrIdx.

C.2 UART Console

Additionally several different UART consoles can be used to provide boot-time-only debug output of the microhypervisor. UART consoles must 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