NOVA Microhypervisor Interface Specification

Udo Steinberg
udo@hypervisor.org

January 18, 2023

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 **unknown**. The microhypervisor cannot ensure that the value does not leak information across protection domain boundaries.

$\nsim$ Indicates that the value of this parameter or field is **undefined**. The microhypervisor ensures that the value does not leak information across protection domain boundaries. Future versions of this specification may define a value 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 preserves the value across hypercalls.

Part I

Introduction

1 System Architecture

The **N**OVA **O**S **V**irtualization **A**rchitecture [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).

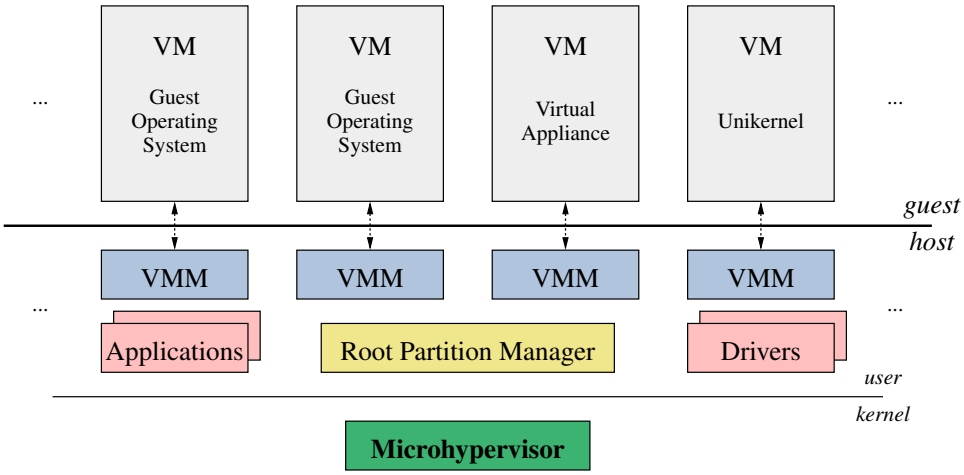


Figure 1.1: System Architecture

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 Capability ($CAP_{OBJ_{PD}}$).
3. Different types of spaces can be created for a PD. These spaces store Capabilities (CAPs) for kernel objects or platform resources that can be accessed by Execution Contexts in that Protection Domain.

The following table shows the types of spaces that exist (✓) for each architecture and the following subsections provide more details for each space.

Architecture	Object Space	Host Space	Guest Space	DMA Space	PIO Space	MSR Space
Arm	✓	✓	✓	✓		
x86	✓	✓	✓	✓	✓	✓

2.1.1 Object Space

1. Only one Object Space (SPC_{OBJ}) can be created per Protection Domain.
2. Access to an Object Space is controlled by an Object Space Capability ($CAP_{OBJ_{OBJ}}$).
3. Each hypercall invoked by a Host Execution Context explicitly specifies an Object Capability Selector to designate the kernel object on which it operates.
4. The Object Capability Selector (SEL_{OBJ}) serves as index into the EC's Object Space and selects a slot that contains either a Null Capability (CAP_0) or an Object Capability (CAP_{OBJ}) that refers to a kernel object with associated access permissions.

2.1.2 Host Space

1. Only one Host Space (SPC_{HST}) can be created per Protection Domain.
2. Access to a Host Space is controlled by a Host Space Capability ($CAP_{OBJ_{HST}}$).
3. Each memory operation issued by a Host Execution Context implicitly uses the page number of the accessed Host-Virtual Address (HVA) as Host Capability Selector: SEL_{HST} = HVA >> 12.
4. The Host Capability Selector (SEL_{HST}) serves as index into the EC's Host Space and selects a slot that contains either a Null Capability (CAP_0) or a Memory Capability (CAP_{MEM}) that refers to a 4 KiB page frame in physical memory with associated access permissions.

2.1.3 Guest Space

1. Multiple Guest Spaces (SPC_{GST}) can be created per Protection Domain.
2. Access to a Guest Space is controlled by a Guest Space Capability ($CAP_{OBJ_{GST}}$).
3. Each memory operation issued by a Guest Execution Context implicitly uses the page number of the accessed Guest-Physical Address (GPA) as Guest Capability Selector: SEL_{GST} = GPA >> 12.
4. The Guest Capability Selector (SEL_{GST}) serves as index into the EC's Guest Space and selects a slot that contains either a Null Capability (CAP_0) or a Memory Capability (CAP_{MEM}) that refers to a 4 KiB page frame in physical memory with associated access permissions.

2.1.4 DMA Space

1. Multiple DMA Spaces (SPC_{DMA}) can be created per Protection Domain.
2. Access to a DMA Space is controlled by a DMA Space Capability ($CAP_{OBJ_{DMA}}$).
3. Each DMA operation issued by a device implicitly uses the page number of the accessed DMA-Virtual Address (DVA) as DMA Capability Selector: SEL_{DMA} = DVA >> 12.
4. The DMA Capability Selector (SEL_{DMA}) serves as index into the device's DMA Space and selects a slot that contains either a Null Capability (CAP_0) or a Memory Capability (CAP_{MEM}) that refers to a 4 KiB page frame in physical memory with associated access permissions.

2.1.5 PIO Space

1. Multiple PIO Spaces (SPC_{PIO}) can be created per Protection Domain.
2. Access to a PIO Space is controlled by a PIO Space Capability ($CAP_{OBJ_{PIO}}$).
3. Each PIO access (`IN/OUT` instruction) issued by an Execution Context (EC) implicitly uses the number of the accessed I/O port as PIO Capability Selector.
4. The PIO Capability Selector (SEL_{PIO}) serves as index into the EC's PIO Space and selects a slot that contains either a Null Capability (CAP_0) or a PIO Capability (CAP_{PIO}) that refers to the I/O port SEL_{PIO} with associated access permissions.

2.1.6 MSR Space

1. Multiple MSR Spaces (SPC_{MSR}) can be created per Protection Domain.
2. Access to an MSR Space is controlled by an MSR Space Capability ($CAP_{OBJ_{MSR}}$).
3. Each MSR access (`RDMSR/WRMSR` instruction) issued by an Execution Context (EC) implicitly uses the number of the accessed MSR as MSR Capability Selector.
4. The MSR Capability Selector (SEL_{MSR}) serves as index into the EC's MSR Space and selects a slot that contains either a Null Capability (CAP_0) or an MSR Capability (CAP_{MSR}) that refers to the MSR SEL_{MSR} with associated access permissions.

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 Capability ($CAP_{OBJ_{EC}}$).
3. An EC is permanently bound to one CPU.
4. An EC contains architecture-dependent state, such as
   - Central Processing Unit (CPU) registers
   - Floating Point Unit (FPU) registers (optionally)

The following subsections provide more details for each type of Execution Context.

2.2.1 Host Execution Context

1. There exist two types of Host Execution Context (EC_{HST}):
   - 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.
2. A Host Execution Context has a UTCB that enables it to perform regular IPC.
3. Upon creation, a Host Execution Context is permanently bound to the following **required** spaces of its PD:
   - The first Object Space*
   - The first Host Space*
   - The first PIO Space†
4. A Host Execution Context cannot be reassigned to any spaces.

2.2.2 Guest Execution Context

1. There exists one type of Guest Execution Context (EC_{GST}):
   - Virtual CPUs – these may have an SC (but no PTs) bound to it.
2. A Guest Execution Context does not have a UTCB.
3. Upon creation, a Guest Execution Context is permanently bound to the following **required** spaces of its PD:
   - The first Object Space*
   - The first Host Space*
4. Upon any architectural/microhypervisor event, a Guest Execution Context may be (re)assigned to the following spaces of its PD, by setting the `SPACES` bit in the MTD and providing selectors that refer to capabilities with `ASSIGN` permission for these spaces in the respective UTCB fields during `ipc_reply`:
   - Any Guest Space
   - Any PIO Space†
   - Any MSR Space†

   The handler of the `STARTUP` event must perform the initial assignment of these spaces.

*Because only one such space can be created for a PD, it is also the only such space.
†Only on architectures, where this type of space exists.

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 Capability ($CAP_{OBJ_{SC}}$).
3. An SC is permanently bound to one 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)
   - Class Of Service (COS)
   - Priority – numerically higher priorities always preempt numerically lower priorities
   - 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 Capability ($CAP_{OBJ_{PT}}$).
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 Capability ($CAP_{OBJ_{SM}}$).

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

Capabilities are a communicable, unforgeable tokens of authority. They consist of a reference to a resource coupled with access permissions. Capabilities are opaque and immutable for applications – they cannot be inspected or modified directly; instead applications refer to a Capability (CAP) via a Capability Selector (SEL).

4.1.1 Null Capability

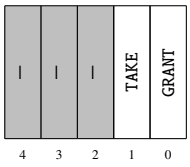
A Null Capability (CAP_0) does not refer to anything and carries no permissions.

4.1.2 Object Capability

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

4.1.2.1 Object Space Capability

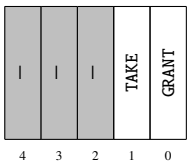
An Object Space Capability ($CAP_{OBJ_{OBJ}}$) refers to an Object Space and carries the following permissions:



`GRANT`	If set, `ctrl_pd` can use that Object Space as destination.
`TAKE`	If set, `ctrl_pd` can use that Object Space as source.

4.1.2.2 Host Space Capability

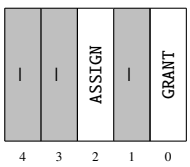
A Host Space Capability ($CAP_{OBJ_{HST}}$) refers to a Host Space and carries the following permissions:



`GRANT`	If set, `ctrl_pd` can use that Host Space as destination.
`TAKE`	If set, `ctrl_pd` can use that Host Space as source.

4.1.2.3 Guest Space Capability

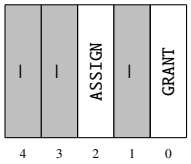
A Guest Space Capability ($CAP_{OBJ_{GST}}$) refers to a Guest Space and carries the following permissions:



`GRANT`	If set, `ctrl_pd` can use that Guest Space as destination.
`ASSIGN`	If set, `ipc_reply` can assign an EC_{GST} to that Guest Space.

4.1.2.4 DMA Space Capability

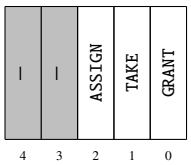
A DMA Space Capability ($CAP_{OBJ_{DMA}}$) refers to a DMA Space and carries the following permissions:



GRANT	If set, `ctrl_pd` can use that DMA Space as destination.
ASSIGN	If set, `assign_dev` can assign a device to that DMA Space.

4.1.2.5 PIO Space Capability

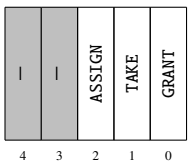
A PIO Space Capability ($CAP_{OBJ_{PIO}}$) refers to a PIO Space and carries the following permissions:



GRANT	If set, `ctrl_pd` can use that PIO Space as destination.
TAKE	If set, `ctrl_pd` can use that PIO Space as source.
ASSIGN	If set, `ipc_reply` can assign an EC_{GST} to that PIO Space.

4.1.2.6 MSR Space Capability

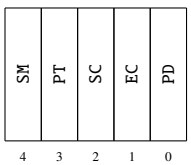
An MSR Space Capability ($CAP_{OBJ_{MSR}}$) refers to an MSR Space and carries the following permissions:



GRANT	If set, `ctrl_pd` can use that MSR Space as destination.
TAKE	If set, `ctrl_pd` can use that MSR Space as source.
ASSIGN	If set, `ipc_reply` can assign an EC_{GST} to that MSR Space.

4.1.2.7 Protection Domain Capability

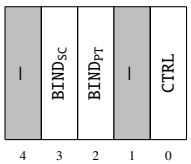
A PD Capability ($CAP_{OBJ_{PD}}$) refers to a Protection Domain (PD) and carries the following permissions:



PD	If set, `create_pd` permitted.
EC	If set, `create_ec` permitted.
SC	If set, `create_sc` permitted.
PT	If set, `create_pt` permitted.
SM	If set, `create_sm` permitted.

4.1.2.8 Execution Context Capability

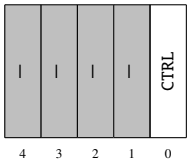
An EC Capability ($CAP_{OBJ_{EC}}$) refers to an Execution Context (EC) and carries the following permissions:



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

4.1.2.9 Scheduling Context Capability

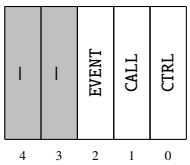
An SC Capability ($CAP_{OBJ_{SC}}$) refers to a Scheduling Context (SC) and carries the following permissions:



CTRL	If set, `ctrl_sc` permitted.

4.1.2.10 Portal Capability

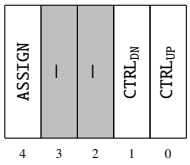
A PT Capability ($CAP_{OBJ_{PT}}$) refers to a Portal (PT) and carries the following permissions:



CTRL	If set, `ctrl_pt` permitted.
CALL	If set, `ipc_call` permitted.
EVENT	If set, delivery of events permitted.

4.1.2.11 Semaphore Capability

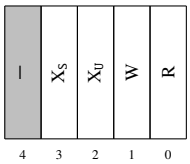
An SM Capability ($CAP_{OBJ_{SM}}$) refers to a Semaphore (SM) and carries the following permissions:



$CTRL_{UP}$	If set, `ctrl_sm` (`Up`) permitted.
$CTRL_{DN}$	If set, `ctrl_sm` (`Down`) permitted.
ASSIGN †	If set, `assign_int` permitted.

4.1.3 Memory Capability

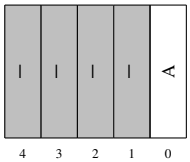
A Memory Capability (CAP_{MEM}) is stored in a Host Space, Guest Space or DMA Space of a PD, refers to a 4 KiB page frame, and carries the following permissions:



R	If set, the page frame is readable.
W	If set, the page frame is writable.
X_U ‡	If set, the page frame is executable (in user mode).
X_S ‡	If set, the page frame is executable (in supervisor mode).

4.1.4 PIO Capability

A PIO Capability (CAP_{PIO}) is stored in a PIO Space of a PD, refers to an I/O port, and carries the following permissions:



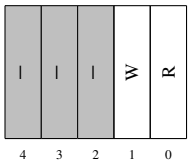
A	If set, the I/O port is accessible (via `IN`/`OUT`).

†This permission bit is only defined for interrupt semaphores.

‡If the hardware supports only combined execute permissions (`X`) for both modes, then $X = X_U \vee X_S$.

4.1.5 MSR Capability

An MSR Capability (CAP_{MSR}) is stored in an MSR Space of a PD, refers to a Model-Specific Register (MSR), and carries the following permissions:



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

4.2 Capability Selector

A Capability Selector (SEL) is an application-visible unsigned number that serves as an index into an Execution Context's space to select a Capability (CAP). The following subsections provide more details.

4.2.1 Object Space Selector

An Object Capability Selector (SEL_{OBJ}) indexes into the Object Space of an Execution Context (EC) and selects a slot that contains either a Null Capability (CAP_0) or an Object Capability (CAP_{OBJ}).

4.2.2 Host Space Selector

A Host Capability Selector (SEL_{HST}) indexes into the Host Space of an Execution Context (EC) and selects a slot that contains either a Null Capability (CAP_0) or a Memory Capability (CAP_{MEM}).

4.2.3 Guest Space Selector

A Guest Capability Selector (SEL_{GST}) indexes into the Guest Space of an Execution Context (EC) and selects a slot that contains either a Null Capability (CAP_0) or a Memory Capability (CAP_{MEM}).

4.2.4 DMA Space Selector

A DMA Capability Selector (SEL_{DMA}) indexes into the DMA Space of an Execution Context (EC) and selects a slot that contains either a Null Capability (CAP_0) or a Memory Capability (CAP_{MEM}).

4.2.5 PIO Space Selector

A PIO Capability Selector (SEL_{PIO}) indexes into the PIO Space of an Execution Context (EC) and selects a slot that contains either a Null Capability (CAP_0) or a PIO Capability (CAP_{PIO}).

4.2.6 MSR Space Selector

An MSR Capability Selector (SEL_{MSR}) indexes into the MSR Space of an Execution Context (EC) and selects a slot that contains either a Null Capability (CAP_0) or an MSR Capability (CAP_{MSR}).

4.3 User Thread Control Block

A User Thread Control Block (UTCB) has a size of one memory page (4 KiB) and is mapped in the Host Space of its associated Host Execution Context. Because a UTCB is allocated and owned by the microhypervisor, it cannot be delegated using `ctrl_pd`.

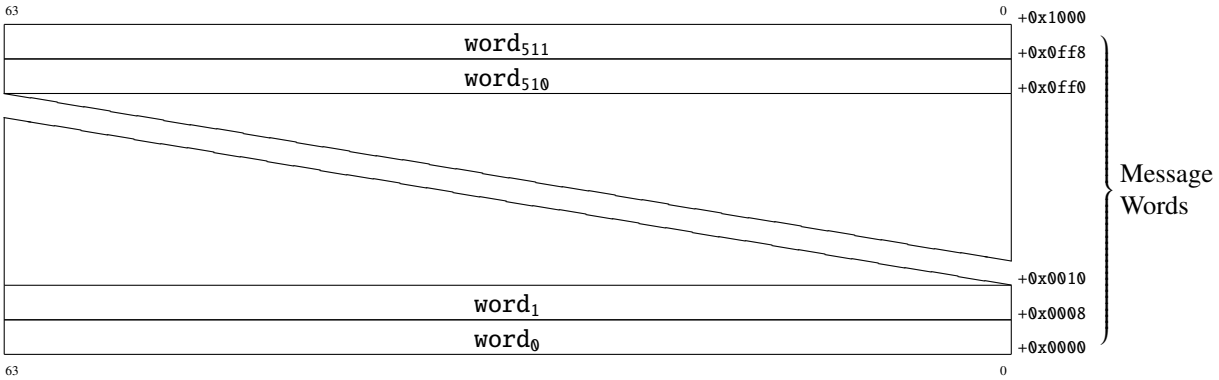
To ensure proper visibility of loads and stores with relaxed memory ordering, application programs are expected to access a UTCB only from the Host Execution Context to which that UTCB is bound.

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:

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).

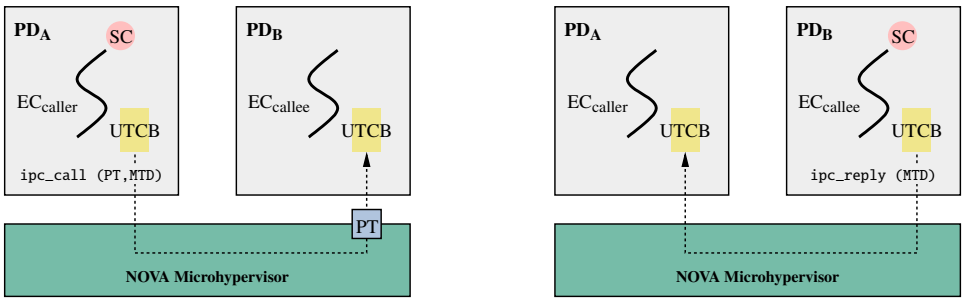


Figure 4.1: Regular IPC

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).

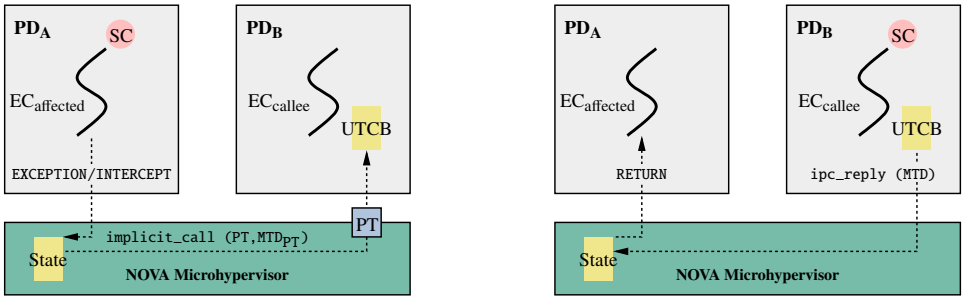
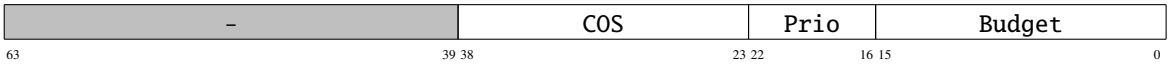


Figure 4.2: Architectural IPC

4.5 Scheduling Context Descriptor

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



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 < COS_{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	MEM_OBJ	Insufficient Memory (Object Creation)
0xb	MEM_CAP	Insufficient Memory (Capability Creation)
≥0xc	*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:

Description:

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

Prior to the hypercall:

- $SPC_{OBJ_{CURRENT}}$[pt] must refer to a PT Capability ($CAP_{OBJ_{PT}}$) 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 invoked 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 ($SC_{CURRENT}$) has been donated to the callee EC upon `ipc_call` and returned back upon `ipc_reply`, thereby accounting the entire handling of the request to $SC_{CURRENT}$.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- $SPC_{OBJ_{CURRENT}}$[pt] did not refer to a PT Capability ($CAP_{OBJ_{PT}}$) 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

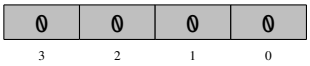
- The callee EC is dead and the operation aborted.

5.2.2 IPC Reply

Parameters:

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

Flags:



Description:

Sends a reply message from $EC_{CURRENT}$ (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 ($SC_{CURRENT}$) that had been donated to the callee EC upon `ipc_call` has been returned back to the caller EC.
- $EC_{CURRENT}$ 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.

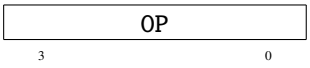
5.3 Object Creation

5.3.1 Create Protection Domain

Parameters:

```
status = create_pd (SEL_OBJ sel,          // Created PD or Space
                    SEL_OBJ pd);          // Owner PD
```

Flags:



Description:

Creates a new Protection Domain (PD) or an empty new space for a Protection Domain.

Prior to the hypercall:

- $SPC_{OBJ_{CURRENT}}$[sel] must refer to a Null Capability (CAP_0).
- $SPC_{OBJ_{CURRENT}}$[pd] must refer to a PD Capability ($CAP_{OBJ_{PD}}$) with permission PD.

If the hypercall completed successfully:

- If **OP=0 (Protection Domain)**:
  - A new Protection Domain (PD) has been created.
  - $SPC_{OBJ_{CURRENT}}$[sel] refers to a $CAP_{OBJ_{PD}}$ for the created Protection Domain with the permissions inherited from $SPC_{OBJ_{CURRENT}}$[pd].
- If **OP=1 (Object Space)**:
  - A new Object Space (SPC_{OBJ}) has been created for the PD referred to by $SPC_{OBJ_{CURRENT}}$[pd].
  - $SPC_{OBJ_{CURRENT}}$[sel] refers to a $CAP_{OBJ_{OBJ}}$ for the created Object Space with all defined permissions.
- If **OP=2 (Host Space)**:
  - A new Host Space (SPC_{HST}) has been created for the PD referred to by $SPC_{OBJ_{CURRENT}}$[pd].
  - $SPC_{OBJ_{CURRENT}}$[sel] refers to a $CAP_{OBJ_{HST}}$ for the created Host Space with all defined permissions.
- If **OP=3 (Guest Space)**:
  - A new Guest Space (SPC_{GST}) has been created for the PD referred to by $SPC_{OBJ_{CURRENT}}$[pd].
  - $SPC_{OBJ_{CURRENT}}$[sel] refers to a $CAP_{OBJ_{GST}}$ for the created Guest Space with all defined permissions.
- If **OP=4 (DMA Space)**:
  - A new DMA Space (SPC_{DMA}) has been created for the PD referred to by $SPC_{OBJ_{CURRENT}}$[pd].
  - $SPC_{OBJ_{CURRENT}}$[sel] refers to a $CAP_{OBJ_{DMA}}$ for the created DMA Space with all defined permissions.
- If **OP=5 (PIO Space)**:
  - A new PIO Space (SPC_{PIO}) has been created for the PD referred to by $SPC_{OBJ_{CURRENT}}$[pd].
  - $SPC_{OBJ_{CURRENT}}$[sel] refers to a $CAP_{OBJ_{PIO}}$ for the created PIO Space with all defined permissions.
- If **OP=6 (MSR Space)**:
  - A new MSR Space (SPC_{MSR}) has been created for the PD referred to by $SPC_{OBJ_{CURRENT}}$[pd].
  - $SPC_{OBJ_{CURRENT}}$[sel] refers to a $CAP_{OBJ_{MSR}}$ for the created MSR Space with all defined permissions.
- The resources for the created object were accounted to the PD referred to by $SPC_{OBJ_{CURRENT}}$[pd].

Status:

SUCCESS

- The hypercall completed successfully.

ABORTED

- If **OP=1**: Attempted creation of more than one Object Space in the PD.
- If **OP=2**: Attempted creation of more than one Host Space in the PD.
- If **OP=5**: Attempted creation of a PIO Space without existing Host Space in the PD.

BAD_CAP

- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[sel] did not refer to a Null Capability (CAP_0).
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd] did not refer to a PD Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) or that capability had insufficient permissions.

BAD_PAR

- The requested operation (OP) is invalid.

BAD_FTR

- The requested space type is not supported by the hardware architecture.

MEM_OBJ

- The Protection Domain referred to by $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd] had insufficient memory resources for object creation.

MEM_CAP

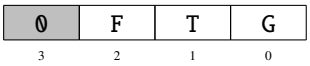
- The Protection Domain to which $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$ belongs had insufficient memory resources for capability creation.

5.3.2 Create Execution Context

Parameters:

```
status = create_ec (SEL_OBJ sel,          // Created EC
                    SEL_OBJ pd,           // Owner PD
                    SEL_HST hvp,          // Host-Virtual Page Number
                    UINT    cpu,          // CPU Number
                    UINT    sp,           // Initial Stack Pointer
                    SEL_EVT evt);         // Event Selector Base
```

Flags:



Description:

Creates a new Execution Context (EC).

Prior to the hypercall:

- $SPC_{OBJ_{CURRENT}}$[sel] must refer to a Null Capability (CAP_0).
- $SPC_{OBJ_{CURRENT}}$[pd] must refer to a PD Capability ($CAP_{OBJ_{PD}}$) with permission EC.
- If **G=0**: All spaces required for a Host Execution Context must have been created for the owner PD.
- If **G=1**: All spaces required for a Guest Execution Context must have been created for the owner PD.

If the hypercall completed successfully:

- If **G=0 (Host Execution Context)**:
  - A new Host Execution Context (EC_{HST}) has been created.
  - The microhypervisor has allocated a UTCB for EC_{HST} and mapped it at HVA hvp << 12.
  - If **T=0 (Local Thread)**: Portals (PTs) may subsequently be bound to EC_{HST}, which will run whenever any of those bound portals is called.
  - If **T=1 (Global Thread)**: EC_{HST} will generate a STARTUP event the first time a Scheduling Context (SC) is bound to it.
- If **G=1 (Guest Execution Context)**:
  - A new Guest Execution Context (EC_{GST}) has been created.
  - The microhypervisor has allocated a vAPIC page for EC_{GST} and mapped it at HVA hvp << 12. On non-Intel architectures, the parameter hvp was ignored.
  - EC_{GST} will generate a STARTUP event the first time a Scheduling Context (SC) is bound to it.
  - If **T=0 (Virtual CPU)**: The virtual CPU uses no time adjustment.
  - If **T=1 (Virtual CPU)**: 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 architecture-specific exception.
- The created EC has its Stack Pointer (SP) set to sp and its Event Selector Base (SEL_{EVT}) set to evt.[†]
- The created EC is permanently bound to the CPU cpu and to the required spaces of the PD referred to by $SPC_{OBJ_{CURRENT}}$[pd].
- The resources for the created EC were accounted to the PD referred to by $SPC_{OBJ_{CURRENT}}$[pd].
- $SPC_{OBJ_{CURRENT}}$[sel] refers to a $CAP_{OBJ_{EC}}$ for the created EC with all defined permissions.

[†]The microhypervisor sets these values only once during EC creation. Subsequently, each Host Execution Context is responsible for maintaining its Stack Pointer (SP) across hypercalls. Applications can use different initial SP or SEL_{EVT} values as a means to identify each Host Execution Context or Guest Execution Context during concurrent STARTUP events.

Status:

SUCCESS

- The hypercall completed successfully.

ABORTED

- Required spaces are missing in the PD referred to by $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd].

BAD_CAP

- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[sel] did not refer to a Null Capability (CAP_0).
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd] did not refer to a PD 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 hardware architecture.

BAD_PAR

- The HVA corresponding to hvp is outside the user-accessible memory range.

MEM_OBJ

- The Protection Domain referred to by $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd] had insufficient memory resources for object creation.

MEM_CAP

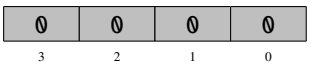
- The Protection Domain to which $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$ belongs had insufficient memory resources for capability creation.

5.3.3 Create Scheduling Context

Parameters:

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

Flags:



Description:

Creates a new Scheduling Context (SC).

Prior to the hypercall:

- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[sel] must refer to a Null Capability (CAP_0).
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd] must refer to a PD Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) with permission SC.
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[ec] must refer to an EC Capability ($\text{CAP}_{\text{OBJ}_{\text{EC}}}$) 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 $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[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 $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd].
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[sel] refers to a $\text{CAP}_{\text{OBJ}_{\text{SC}}}$ for the created SC with all defined permissions.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[sel] did not refer to a Null Capability (CAP_0).
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd] did not refer to a PD Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) or that capability had insufficient permissions.
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[ec] did not refer to an EC Capability ($\text{CAP}_{\text{OBJ}_{\text{EC}}}$) or that capability had insufficient permissions.
- Binding the SC to the EC failed, e.g. because the EC is a local thread.

BAD_PAR

- At least one SCD field in scd was invalid.

MEM_OBJ

- The Protection Domain referred to by $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd] had insufficient memory resources for object creation.

MEM_CAP

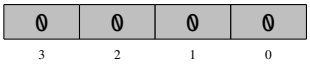
- The Protection Domain to which $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$ belongs had insufficient memory resources for capability creation.

5.3.4 Create Portal

Parameters:

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

Flags:



Description:

Creates a new Portal (PT).

Prior to the hypercall:

- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[sel] must refer to a Null Capability (CAP_0).
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd] must refer to a PD Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) with permission PT.
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[ec] must refer to an EC Capability ($\text{CAP}_{\text{OBJ}_{\text{EC}}}$) with permission BIND_{PT}.

If the hypercall completed successfully:

- A new Portal (PT) has been created.
- The created PT is bound to the EC referred to by $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[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 $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd].
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[sel] refers to a $\text{CAP}_{\text{OBJ}_{\text{PT}}}$ for the created PT with all defined permissions.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[sel] did not refer to a Null Capability (CAP_0).
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd] did not refer to a PD Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) or that capability had insufficient permissions.
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[ec] did not refer to an EC 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 thread.

MEM_OBJ

- The Protection Domain referred to by $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pd] had insufficient memory resources for object creation.

MEM_CAP

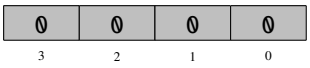
- The Protection Domain to which $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$ belongs had insufficient memory resources for capability creation.

5.3.5 Create Semaphore

Parameters:

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

Flags:



Description:

Creates a new Semaphore (SM).

Prior to the hypercall:

- $SPC_{OBJ_{CURRENT}}$[sel] must refer to a Null Capability (CAP_0).
- $SPC_{OBJ_{CURRENT}}$[pd] must refer to a PD Capability ($CAP_{OBJ_{PD}}$) 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 $SPC_{OBJ_{CURRENT}}$[pd].
- $SPC_{OBJ_{CURRENT}}$[sel] refers to a $CAP_{OBJ_{SM}}$ for the created SM with all defined permissions.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- $SPC_{OBJ_{CURRENT}}$[sel] did not refer to a Null Capability (CAP_0).
- $SPC_{OBJ_{CURRENT}}$[pd] did not refer to a PD Capability ($CAP_{OBJ_{PD}}$) or that capability had insufficient permissions.

MEM_OBJ

- The Protection Domain referred to by $SPC_{OBJ_{CURRENT}}$[pd] had insufficient memory resources for object creation.

MEM_CAP

- The Protection Domain to which $SPC_{OBJ_{CURRENT}}$ belongs had insufficient memory resources for capability creation.

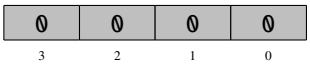
5.4 Object Control

5.4.1 Control Protection Domain

Parameters:

```
status = ctrl_pd (SEL_OBJ src,          // SRC Space
                  SEL_OBJ dst,          // DST Space
                  SEL     ssb,          // SRC Selector Base
                  SEL     dsb,          // DST Selector Base
                  UINT    ord,          // Order
                  UINT    pmm,          // Permission Mask
                  MAD     mad);         // Memory Attribute Descriptor
```

Flags:



Description:

Takes capabilities from the source space and grants them to the destination space and thereby optionally reduces the permissions of the destination capabilities.

Prior to the hypercall:

- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[src] must refer to a CAP_{OBJ} for the source space with permission TAKE.
- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[dst] must refer to a CAP_{OBJ} for the destination space with permission GRANT.
- Capability Selectors ssb and dsb must be order-aligned: ssb≡0 (mod 2^{ord}) and dsb≡0 (mod 2^{ord}).
- Capability Selectors ssb and dsb must be equal if src refers to a PIO Space or an MSR Space.

If the hypercall completed successfully:

- If both src and dst refer to Object Spaces:
  - All CAP_0 and CAP_{OBJ} from SEL_{OBJ} range src[ssb...ssb+2^{ord}-1] were delegated to SEL_{OBJ} range dst[dsb...dsb+2^{ord}-1].
  - Any pre-existing CAP_{OBJ} in the SEL_{OBJ} range dst[dsb...dsb+2^{ord}-1] were revoked.
  - The parameter mad was ignored.
- If src refers to a Host Space and dst refers to a Host Space or Guest Space or DMA Space:
  - All CAP_0 and CAP_{MEM} from SEL_{HST} range src[ssb...ssb+2^{ord}-1] were delegated to SEL range dst[dsb...dsb+2^{ord}-1].
  - Any pre-existing CAP_{MEM} in the SEL range dst[dsb...dsb+2^{ord}-1] were revoked.
  - If src refers to the NOVA Host Space, then the source SEL_{HST} are **physical** page numbers and the memory attributes of each destination CAP_{MEM} were *set* according to mad.

    Otherwise, the source SEL_{HST} are **virtual** page numbers and the memory attributes of each destination CAP_{MEM} were *inherited* from the respective source CAP_{MEM}, i.e. the parameter mad was ignored.
- If both src and dst refer to PIO Spaces:
  - All CAP_0 and CAP_{PIO} from SEL_{PIO} range src[ssb...ssb+2^{ord}-1] were delegated to SEL_{PIO} range dst[dsb...dsb+2^{ord}-1].
  - Any pre-existing CAP_{PIO} in the SEL_{PIO} range dst[dsb...dsb+2^{ord}-1] were revoked.
  - The parameter mad was ignored.

- If both `src` and `dst` refer to MSR Spaces:
  - All CAP_0 and CAP_{MSR} from SEL_{MSR} range `src`[`ssb`...`ssb`+2^{ord}−1] were delegated to SEL_{MSR} range `dst`[`dsb`...`dsb`+2^{ord}−1].
  - Any pre-existing CAP_{MSR} in the SEL_{MSR} range `dst`[`dsb`...`dsb`+2^{ord}−1] were revoked.
  - The parameter `mad` was 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 a Null Capability (CAP_0).
- The resources for storing the granted capabilities were accounted to the PD to which the space referred to by $SPC_{OBJ_{CURRENT}}$[`dst`] belongs.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- $SPC_{OBJ_{CURRENT}}$[`src`] or $SPC_{OBJ_{CURRENT}}$[`dst`] did not refer to compatible space capabilities or had insufficient permissions.

BAD_PAR

- Capability Selector `ssb` or `dsb` was not order-aligned.
- Capability Selector `ssb`+2^{ord}−1 or `dsb`+2^{ord}−1 was larger than the maximum selector number.
- If `src` refers to a PIO Space or an MSR Space: Capability Selectors `ssb` and `dsb` were not equal.
- If `src` refers to the NOVA Host Space: At least one MAD field in `mad` was invalid.

MEM_CAP

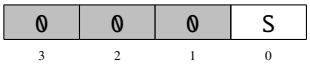
- The PD to which the space referred to by $SPC_{OBJ_{CURRENT}}$[`dst`] belongs had insufficient memory resources for allocating the storage required for granting all destination capabilities. This constitutes a partial failure of the operation, because all destination capabilities up to the first allocation failure have been granted.

5.4.2 Control Execution Context

Parameters:

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

Flags:



Description:

Prior to the hypercall:

- $SPC_{OBJ_{CURRENT}}$[ec] must refer to an EC Capability ($CAP_{OBJ_{EC}}$) with permission CTRL.

If the hypercall completed successfully:

- The EC referred to by $SPC_{OBJ_{CURRENT}}$[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

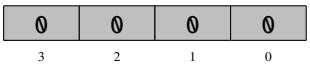
- $SPC_{OBJ_{CURRENT}}$[ec] did not refer to an EC Capability ($CAP_{OBJ_{EC}}$) or that capability had insufficient permissions.

5.4.3 Control Scheduling Context

Parameters:

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

Flags:



Description:

Prior to the hypercall:

- $SPC_{OBJ_{CURRENT}}$[sc] must refer to an SC Capability ($CAP_{OBJ_{SC}}$) 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 $SPC_{OBJ_{CURRENT}}$[sc].

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

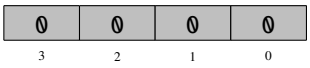
- $SPC_{OBJ_{CURRENT}}$[sc] did not refer to an SC Capability ($CAP_{OBJ_{SC}}$) or that capability had insufficient permissions.

5.4.4 Control Portal

Parameters:

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

Flags:



Description:

Prior to the hypercall:

- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pt] must refer to a PT Capability ($\text{CAP}_{\text{OBJ}_{\text{PT}}}$) with permission CTRL.

If the hypercall completed successfully:

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

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- $\text{SPC}_{\text{OBJ}_{\text{CURRENT}}}$[pt] did not refer to a PT Capability ($\text{CAP}_{\text{OBJ}_{\text{PT}}}$) or that capability had insufficient permissions.

5.4.5 Control Semaphore

Parameters:

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

Flags:

Description:

Prior to the hypercall:

- If **D=0 (Semaphore Up)**:
  - $SPC_{OBJ_{CURRENT}}$[sm] must refer to an SM Capability ($CAP_{OBJ_{SM}}$) with permission $CTRL_{UP}$.
- If **D=1 (Semaphore Down)**:
  - $SPC_{OBJ_{CURRENT}}$[sm] must refer to an SM Capability ($CAP_{OBJ_{SM}}$) 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 $EC_{CURRENT}$ on the semaphore. If the timeout value was non-zero, $EC_{CURRENT}$ 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

- $SPC_{OBJ_{CURRENT}}$[sm] did not refer to an SM Capability ($CAP_{OBJ_{SM}}$) 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`.

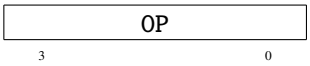
5.5 Platform Management

5.5.1 Control Hardware

Parameters:

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

Flags:



Description:

Modifies the platform hardware configuration or power management state.

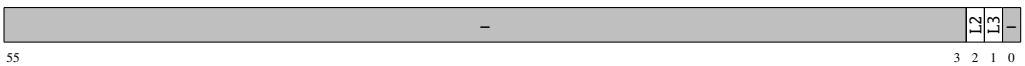
Prior to the hypercall:

- The hypercall must be invoked by an Execution Context in 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:

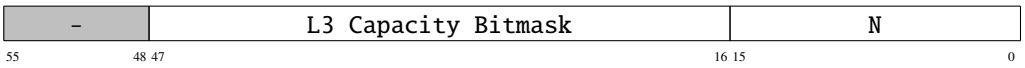
S	A	B	Shallow	Description
0x1	_S1[0]	_S1[1]	✓	S1: Stop Grant
0x2	_S2[0]	_S2[1]	✓	S2: Power-On Suspend
0x3	_S3[0]	_S3[1]	✓	S3: Suspend to RAM
0x4	_S4[0]	_S4[1]	×	S4: Suspend to Disk
0x5	_S5[0]	_S5[1]	×	S5: Soft Off
0x7	0x0	0x0	×	Platform Reset

  - 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:

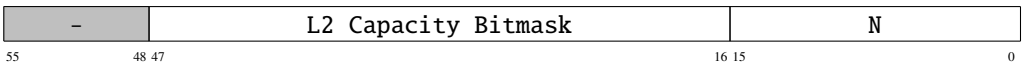


  - Only Code and Data Prioritization (CDP) settings supported by the ambient CPU are valid.
    - The L3 bit disables (0) or enables (1) CDP_{L3} on the ambient CPU.
    - The L2 bit disables (0) or enables (1) CDP_{L2} on the ambient CPU.
  - CAT/CDP or MBA settings cannot be configured for a CPU until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

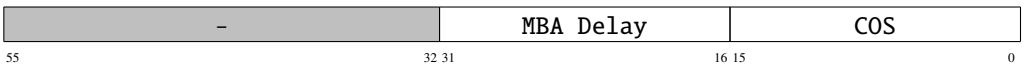
- If **OP=5 (CAT/CDP L3 Capacity Bitmask)**:
  - The descriptor `desc` uses the following encoding:



  - 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 $CDP_{L3\text{-}Data}$ Capacity Bitmask for a COS, use `N=(COS<<1)`.
      - To configure the $CDP_{L3\text{-}Code}$ 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:



  - 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 $CDP_{L2\text{-}Data}$ Capacity Bitmask for a COS, use `N=(COS<<1)`.
      - To configure the $CDP_{L2\text{-}Code}$ 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:



  - 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 invoked 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:

- $SPC_{OBJ_{CURRENT}}$[sm] must refer to an SM Capability ($CAP_{OBJ_{SM}}$) with permission ASSIGN.
- $CAP_{OBJ_{SM}}$ must refer to an interrupt semaphore and thereby designates the interrupt.

If the hypercall completed successfully:

- The interrupt referred to by $SPC_{OBJ_{CURRENT}}$[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

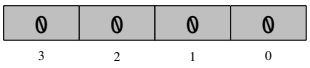
- $SPC_{OBJ_{CURRENT}}$[sm] did not refer to an SM Capability ($CAP_{OBJ_{SM}}$) or that capability had insufficient permissions.
- $CAP_{OBJ_{SM}}$ did not refer to an interrupt semaphore.

5.5.3 Assign Device

Parameters:

```
status = assign_dev (SEL_OBJ   dma,        // DMA Space
                     UINT      smmu,       // SMMU Physical Address (Page Number)
                     DAD       dad);       // Device Assignment Descriptor
```

Flags:



Description:

Assigns a device to the specified DMA Space.

Prior to the hypercall:

- The hypercall must be invoked by an Execution Context in the Root Protection Domain (PD_{ROOT}).
- $SPC_{OBJ_{CURRENT}}$[dma] must refer to a DMA Space Capability ($CAP_{OBJ_{DMA}}$) with permission ASSIGN.
- The parameter smmu designates the SMMU/IOMMU via its physical address.
- DAD designates the device and SMMU resources to use for managing that device.

If the hypercall completed successfully:

- The device designated by DAD has been assigned to the DMA Space referred to by $SPC_{OBJ_{CURRENT}}$[dma].
- 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 invoked from the Root Protection Domain (PD_{ROOT}).

BAD_DEV

- The parameter smmu did not refer to the physical address of an SMMU device.

BAD_CAP

- $SPC_{OBJ_{CURRENT}}$[dma] did not refer to a DMA Space Capability ($CAP_{OBJ_{DMA}}$) or that capability had insufficient permissions.

BAD_PAR

- The parameter dad was invalid.

6 Booting

6.1 Microhypervisor

6.1.1 NOVA ELF Image

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 0x000000000fff000 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. $\texttt{PhysAddr}_{\text{NEW}} = \texttt{PhysAddr}_{\text{ELF}} \pm n \times 2\ \texttt{MiB}$.
- 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 NOVA Object Space

The NOVA Object Space ($SPC_{OBJ_{NOVA}}$) contains the following CAP_{OBJ}:

SEL_{OBJ}	Capability Type	Capability Resource	Capability Permissions
SEL_{NUM}-1	$CAP_{OBJ_{SM}}$	Console Semaphore	All defined permissions
SEL_{NUM}-2	$CAP_{OBJ_{OBJ}}$	NOVA Object Space	TAKE
SEL_{NUM}-3	$CAP_{OBJ_{HST}}$	NOVA Host Space	TAKE
SEL_{NUM}-4	$CAP_{OBJ_{PIO}}$	NOVA PIO Space[†]	TAKE
SEL_{NUM}-5	$CAP_{OBJ_{MSR}}$	NOVA MSR Space[†]	TAKE
SEL_{NUM}-6	$CAP_{OBJ_{OBJ}}$	Root Object Space	All defined permissions
SEL_{NUM}-7	$CAP_{OBJ_{HST}}$	Root Host Space	All defined permissions
SEL_{NUM}-8	$CAP_{OBJ_{PIO}}$	Root PIO Space[†]	All defined permissions
2^{16}...2^{16}+INT_{PIN}+INT_{MSI}-1	$CAP_{OBJ_{SM}}$	Interrupt Semaphores	ASSIGN, $CTRL_{DN}$
0...CPU_{NUM}-1	$CAP_{OBJ_{SC}}$	Idle Scheduling Contexts	CTRL

Using $SPC_{OBJ_{NOVA}}$ as source space for the ctrl_pd hypercall facilitates the delegation of CAP_{OBJ} from the NOVA Object Space to another Object Space.

6.1.3 NOVA Host Space

In the NOVA Host Space ($SPC_{HST_{NOVA}}$), SEL_{HST} N refers to

- CAP_0 for memory protected by the NOVA microhypervisor (Arm, x86).
- CAP_{MEM} for the 4 KiB page frame at **physical** address N << 12 with all defined permissions otherwise.

Using $SPC_{HST_{NOVA}}$ as source space for the ctrl_pd hypercall facilitates the delegation of CAP_{MEM} from the NOVA Host Space to another Host Space, Guest Space or DMA Space.

6.1.4 NOVA PIO Space

In the NOVA PIO Space ($SPC_{PIO_{NOVA}}$), SEL_{PIO} N refers to

- CAP_0 for an I/O port protected by the NOVA microhypervisor (x86).
- CAP_{PIO} for the I/O port number N with certain permissions otherwise (x86).

Using $SPC_{PIO_{NOVA}}$ as source space for the ctrl_pd hypercall facilitates the delegation of CAP_{PIO} from the NOVA PIO Space to another PIO Space.

6.1.5 NOVA MSR Space

In the NOVA MSR Space ($SPC_{MSR_{NOVA}}$), SEL_{MSR} N refers to

- CAP_0 for an MSR protected by the NOVA microhypervisor (x86).
- CAP_{MSR} for the MSR number N with certain permissions otherwise (x86).

Using $SPC_{MSR_{NOVA}}$ as source space for the ctrl_pd hypercall facilitates the delegation of CAP_{MSR} from the NOVA MSR Space to another MSR Space.

[†]Only on architectures, where this type of space exists, otherwise Null Capability (CAP_0).

6.2 Root Protection Domain

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

- Root Protection Domain (PD_{ROOT})
  - with Root Object Space ($SPC_{OBJ_{ROOT}}$)
  - with Root Host Space ($SPC_{HST_{ROOT}}$)
  - with Root PIO Space ($SPC_{PIO_{ROOT}}$)[†]
- Root Execution Context (EC_{ROOT}) on CPU_{BSP}
  - bound to $SPC_{OBJ_{ROOT}}$, $SPC_{HST_{ROOT}}$, $SPC_{PIO_{ROOT}}$[†]
  - with $SEL_{EVT} = 0$
- Root Scheduling Context (SC_{ROOT}) on CPU_{BSP}
  - bound to EC_{ROOT}
  - with COS = 0
  - with Priority = highest priority
  - with Budget = 1000 ms

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 Root ELF Image

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
                 0x000000000000a75  0x000000000000a75  R E    0x1000
  LOAD           0x0000000000001000 0x0000000010001000 0x0000000010001000
                 0x000000000000f004 0x000000000000f004 RW     0x1000
```

Prior to the Root Execution Context (EC_{ROOT}) invoking the entry point of the Root Protection Domain (PD_{ROOT}) ELF image, the NOVA microhypervisor sets up the spaces for PD_{ROOT} as described in the following subsections.

[†] Only on architectures, where this type of space exists.
* This is the address in physical memory at which the bootloader has placed the ELF image.

6.2.2 Root Object Space

The Root Object Space ($SPC_{OBJ_{ROOT}}$) contains the following initial CAP_{OBJ}:

SEL_{OBJ}	Capability Type	Capability Resource	Capability Permissions
SEL_{NUM}-1	$CAP_{OBJ_{OBJ}}$	NOVA Object Space	TAKE
SEL_{NUM}-2	$CAP_{OBJ_{OBJ}}$	Root Object Space	All defined permissions
SEL_{NUM}-3	$CAP_{OBJ_{PD}}$	Root Protection Domain	All defined permissions
SEL_{NUM}-4	$CAP_{OBJ_{EC}}$	Root Execution Context	All defined permissions
SEL_{NUM}-5	$CAP_{OBJ_{SC}}$	Root Scheduling Context	All defined permissions

All other SEL_{OBJ} in $SPC_{OBJ_{ROOT}}$ initially refer to a Null Capability (CAP_0).

6.2.3 Root Host Space

ELF Program Segments

The microhypervisor maps the Root Protection Domain (PD_{ROOT}) into the Root Host Space 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 Root Host 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 Root Host Space 4 KiB below the address of the Hypervisor Information Page.

All other SEL_{HST} in $SPC_{HST_{ROOT}}$ initially refer to a Null Capability (CAP_0).

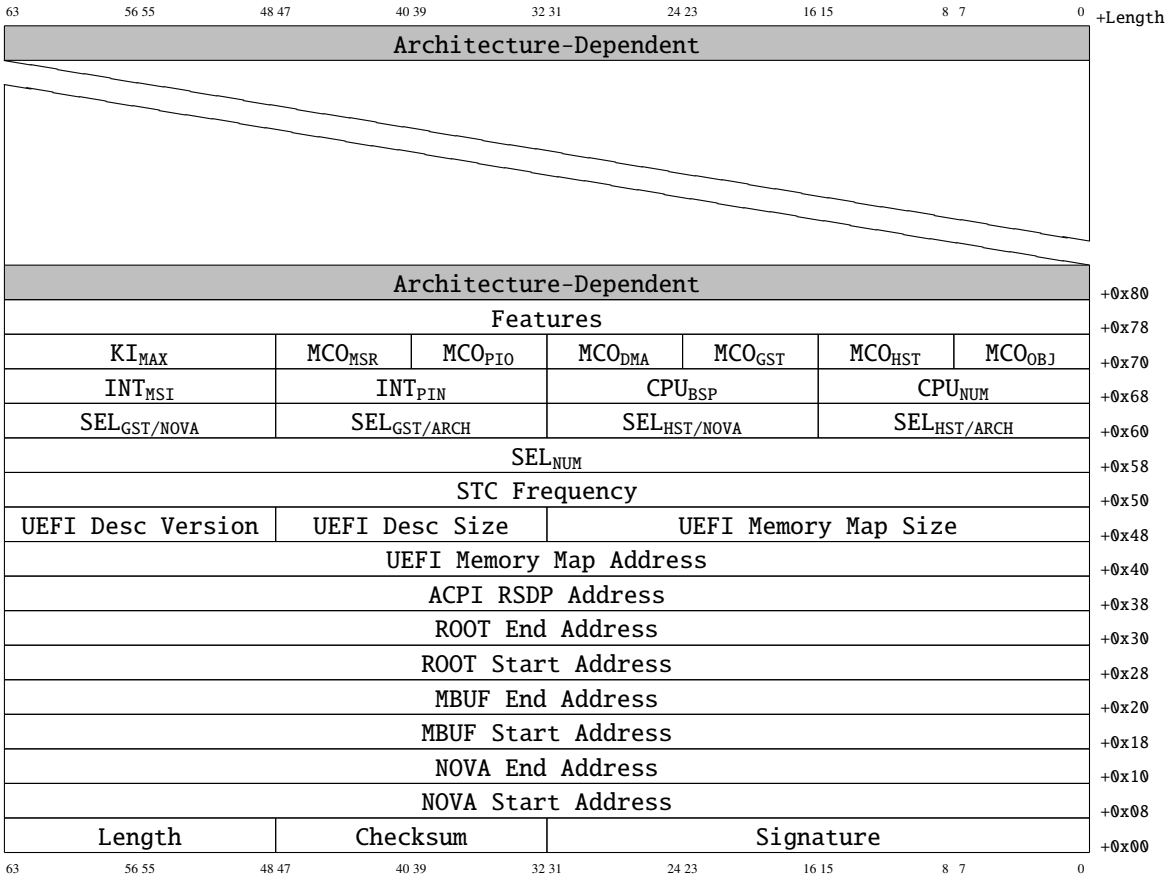
6.2.4 Root PIO Space

All SEL_{PIO} in $SPC_{PIO_{ROOT}}$ initially refer to a Null Capability (CAP_0).

6.3 Hypervisor Information Page

The Hypervisor Information Page (HIP) is mapped in the Host Space of the Root Protection Domain and the initial Stack Pointer (SP) of the Root Execution Context points to the HIP. Because the HIP is allocated and owned by the microhypervisor, it cannot be delegated using `ctrl_pd`.

The HIP conveys information about the platform and configuration and has the following layout:



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.

$SEL_{HST/ARCH}$
Number of Capability Selectors required for handling architectual host events. (Arm, x86)

$SEL_{HST/NOVA}$
Number of additional Capability Selectors required for handling microhypervisor host events. (Arm, x86)

$SEL_{GST/ARCH}$
Number of Capability Selectors required for handling architectual guest events. (Arm, x86)

$SEL_{GST/NOVA}$
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.

MCO_{OBJ}
Maximum contiguous order that avoids partial failures during capability updates in SPC_{OBJ}.

MCO_{HST}
Maximum contiguous order that avoids partial failures during capability updates in SPC_{HST}.

MCO_{GST}
Maximum contiguous order that avoids partial failures during capability updates in SPC_{GST}.

MCO_{DMA}
Maximum contiguous order that avoids partial failures during capability updates in SPC_{DMA}.

MCO_{PIO}
Maximum contiguous order that avoids partial failures during capability updates in SPC_{PIO}.

MCO_{MSR}
Maximum contiguous order that avoids partial failures during capability updates in SPC_{MSR}.

KI_{MAX}
Maximum key identifier value.

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 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 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 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_{ROOT}):

Register	Value / Description
IP	HVA of the Root Protection Domain (PD_{ROOT}) ELF image entry point
SP	HVA of the Hypervisor Information Page (HIP)
X0	X0 at boot time [†]
X1	X1 at boot time [†]
X2	X2 at boot time [†]
Other	$\nsim$

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

7.2 Protected Resources

Certain resources protected by the NOVA microhypervisor cannot be delegated and therefore remain inaccessible to user-mode components. The following subsections enumerate these protected resources.

7.2.1 Memory

The following physical memory regions are protected:

- 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

7.4.1 Host-Virtual Addresses

Depending on the MMU features supported by the hardware platform, the Host-Virtual Address (HVA) range accessible to Host Execution Contexts is $0 \leq HVA < 2^H$ with $32 \leq H \leq 52$.

The Root Protection Domain (PD_{ROOT}) should determine 2^H based on the virtual address of the HIP.

7.4.2 Guest-Physical Addresses

Depending on the MMU features supported by the hardware platform, the Guest-Physical Address (GPA) range accessible to Guest Execution Contexts is $0 \leq GPA < 2^G$ with $32 \leq G \leq 52$ and $G = H$.

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

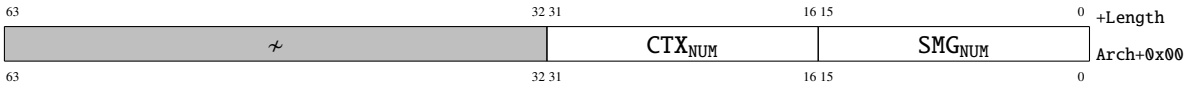
SEL_{OBJ}	Event
SEL_{EVT} + SEL_{ARCH} + 0x0	Startup
SEL_{EVT} + SEL_{ARCH} + 0x1	Recall
SEL_{EVT} + SEL_{ARCH} + 0x2	Virtual Timer

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

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

7.7 Architecture-Dependent Structures

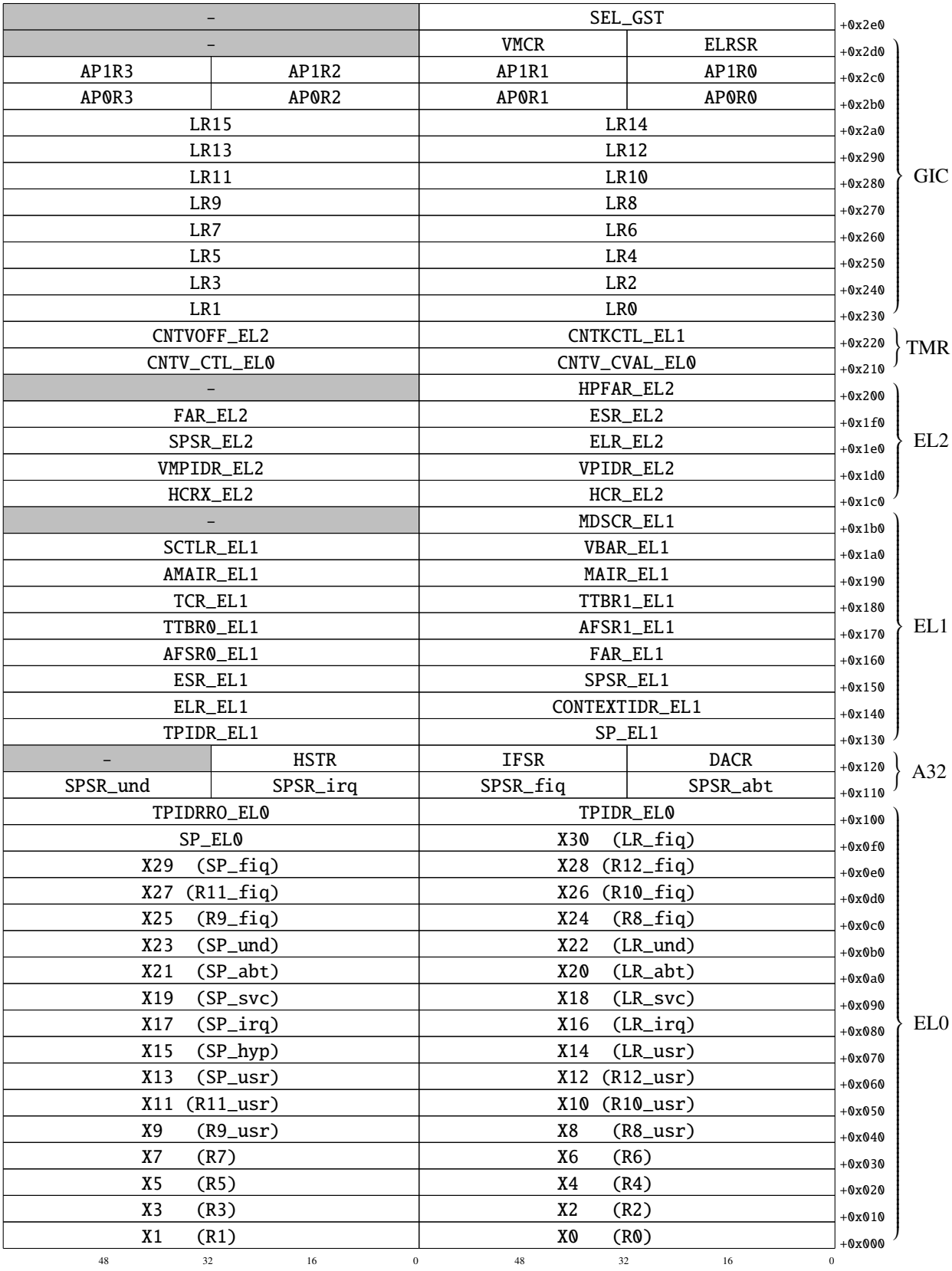
7.7.1 Hypervisor Information Page



SMG_{NUM}
Total number of Stream Mapping Groups (SMGs).

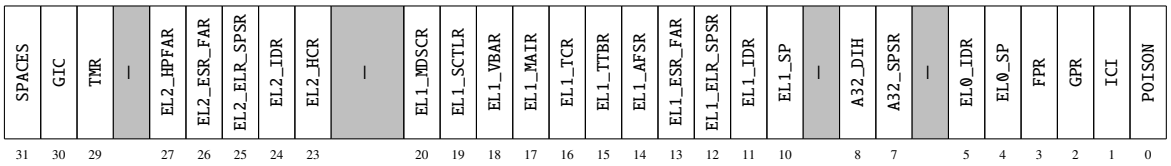
CTX_{NUM}
Total number of Translation Contexts (CTXs).

7.7.2 User Thread Control Block



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:



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 Execution Context State	Guest Execution Context State
POISON	w	Kills the EC_{HST}	Kills the EC_{GST}
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
SPACES	w	-	SEL_GST

*Only the condition flags are writable.
†Only affects a VIPT instruction cache of the local CPU. Has no effect on PIPT instruction caches, data caches, or caches of other CPUs.

7.7.4 Memory Attribute Descriptor

The Memory Attribute Descriptor (MAD) describes page attributes for a CAP_{MEM} delegation from $SPC_{HST_{NOVA}}$.

The fields are defined as follows:

CA

Specifies the cacheability attributes of the page according to the following table:

Encoding	Cacheability	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

SH

Specifies the shareability attributes of the page according to the following table:

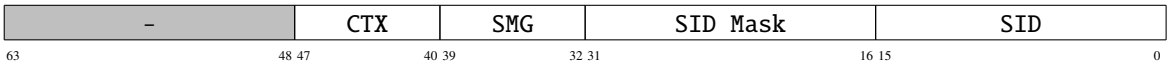
Encoding	Shareability	Description
0x0	NONE	Not Shareable
0x1	-	*reserved*
0x2	OUTER	Outer Shareable
0x3	INNER	Inner Shareable

Please refer to [3] for the architectural behavior of these attributes.

7.7.5 Device Assignment Descriptor

The Device Assignment Descriptor (DAD) describes a device to be assigned to a DMA Space (SPC_{DMA}).

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



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 $< SMG_{NUM}$.

CTX

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

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. A Host 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.

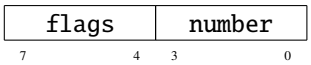


Figure 7.1: Hypercall Identifier

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

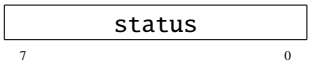


Figure 7.2: Status Code

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	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	ipc_reply →	X0	pid
$mtd_{[31-0]}$	X1		X1	$mtd_{[31-0]}$
-	IP	svc #0	IP	Portal IP

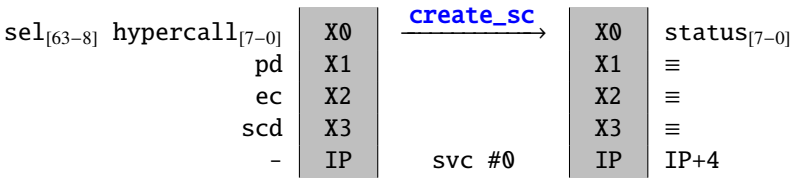
Create Protection Domain

$sel_{[63-8]}$ $hypercall_{[7-0]}$	X0	create_pd →	X0	$status_{[7-0]}$
pd	X1		X1	≡
-	IP	svc #0	IP	IP+4

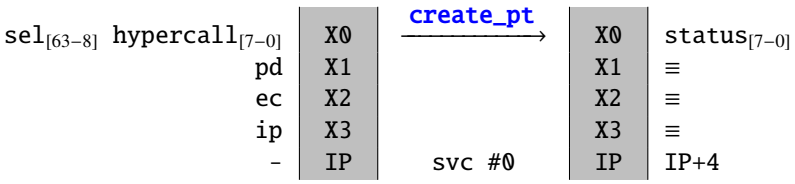
Create Execution Context

$sel_{[63-8]}$ $hypercall_{[7-0]}$	X0	create_ec →	X0	$status_{[7-0]}$
pd	X1		X1	≡
$hvp_{[63-12]}$ $cpu_{[11-0]}$	X2		X2	≡
sp	X3		X3	≡
evt	X4		X4	≡
-	IP	svc #0	IP	IP+4

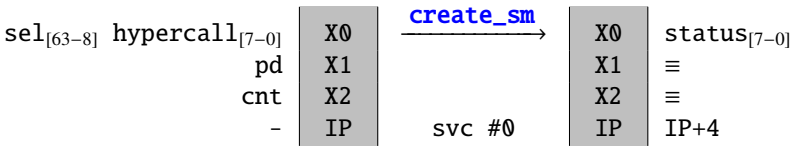
Create Scheduling Context



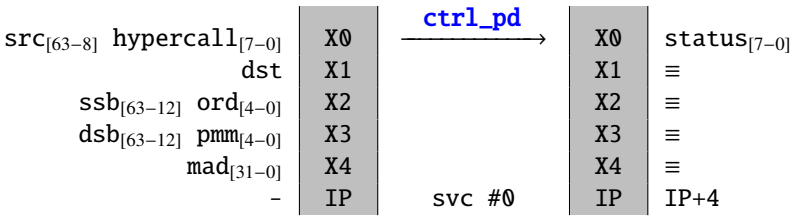
Create Portal



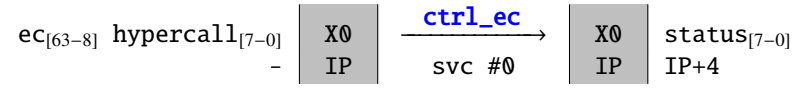
Create Semaphore



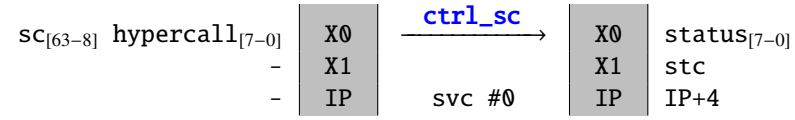
Control Protection Domain



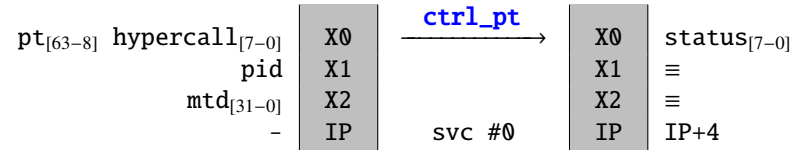
Control Execution Context



Control Scheduling Context



Control Portal



Control Semaphore

$sm_{[63-8]}$	$hypercall_{[7-0]}$	X0	ctrl_sm ⟶	X0	$status_{[7-0]}$
	stc	X1		X1	≡
	-	IP	svc #0	IP	IP+4

Control Hardware

$desc_{[63-8]}$	$hypercall_{[7-0]}$	X0	ctrl_hw ⟶	X0	$status_{[7-0]}$
	-	IP	svc #0	IP	IP+4

Assign Interrupt

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

Assign Device

$dma_{[63-8]}$	$hypercall_{[7-0]}$	X0	assign_dev ⟶	X0	$status_{[7-0]}$
	$smmu_{[63-12]}$	X1		X1	≡
	dad	X2		X2	≡
	-	IP	svc #0	IP	IP+4

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

		proxy_smc →		
identifier$_{[31-0]}$	X0		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 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 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`	HVA of the Root Protection Domain (PD_{ROOT}) ELF image entry point
`RSP`	HVA 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

Certain resources protected by the NOVA microhypervisor cannot be delegated and therefore remain inaccessible to user-mode components. The following subsections enumerate these protected resources.

8.2.1 Memory

The following physical memory regions are protected:

- 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 Ports

The CAP_{PIO} column in the table below details the permissions granted for I/O ports by the NOVA PIO Space. Each I/O port is either unprotected or fully protected.

I/O Port	CAP_{PIO}	Exception or I/O Exit	Enumeration	VMM Handling

`PM1a_CNT`
`PM1b_CNT`
`PM2_CNT`
`SMI_CMD`
	CAP_0	Always	ACPI FADT	Emulate value and side effects

Other	`A`	If `A` is not set	–	Passthrough or Emulate

8.2.3 Model-Specific Registers

The CAP_{MSR} column in the table below details the permissions granted for MSRs by the NOVA MSR Space. Each MSR is either unprotected or write-protected or fully protected.

Model-Specific Register	CAP_{MSR}	RDMSR Exit	WRMSR Exit	VMM Handling

`IA32_SYSENTER_CS`
`IA32_SYSENTER_ESP`
`IA32_SYSENTER_EIP`
`IA32_PAT`
`IA32_EFER`
`IA32_FS_BASE`
`IA32_GS_BASE`
`IA32_KERNEL_GS_BASE`
	`RW`	If `R` is not set	If `W` is not set	Read/Write effective value via UTCB

`IA32_XSS`
`IA32_STAR`
`IA32_LSTAR`
`IA32_FMASK`
`IA32_TSC_AUX`
	`R`	If `R` is not set	Always	

Other	CAP_0	Always		Emulate value and side effects

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

8.4.1 Host-Virtual Addresses

Depending on the MMU features supported by the hardware platform, the Host-Virtual Address (HVA) range accessible to Host Execution Contexts is $0 \leq HVA < 2^H$ with $47 \leq H \leq 56$.

The Root Protection Domain (PD_{ROOT}) should determine 2^H based on the virtual address of the HIP.

8.4.2 Guest-Physical Addresses

Depending on the MMU features supported by the hardware platform, the Guest-Physical Address (GPA) range accessible to Guest Execution Contexts is $0 \leq GPA < 2^G$ with $48 \leq G \leq 57$ and $G = H+1$.

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 Capability ($CAP_{OBJ_{PT}}$) with permission `EVENT` that is bound to an EC on the same CPU as the affected EC, otherwise the affected EC is killed.

8.6.1 Architectural Events

Host Exceptions

SEL_{OBJ}	Exception	SEL_{OBJ}	Exception
SEL_{EVT} + 0x00	#DE	SEL_{EVT} + 0x10	#MF
SEL_{EVT} + 0x01	#DB	SEL_{EVT} + 0x11	#AC
SEL_{EVT} + 0x02	reserved	SEL_{EVT} + 0x12	#MC*
SEL_{EVT} + 0x03	#BP	SEL_{EVT} + 0x13	#XM
SEL_{EVT} + 0x04	#OF	SEL_{EVT} + 0x14	#VE
SEL_{EVT} + 0x05	#BR	SEL_{EVT} + 0x15	#CP
SEL_{EVT} + 0x06	#UD	SEL_{EVT} + 0x16	reserved
SEL_{EVT} + 0x07	#NM*	SEL_{EVT} + 0x17	reserved
SEL_{EVT} + 0x08	#DF*	SEL_{EVT} + 0x18	reserved
SEL_{EVT} + 0x09	reserved	SEL_{EVT} + 0x19	reserved
SEL_{EVT} + 0x0a	#TS*	SEL_{EVT} + 0x1a	reserved
SEL_{EVT} + 0x0b	#NP	SEL_{EVT} + 0x1b	reserved
SEL_{EVT} + 0x0c	#SS	SEL_{EVT} + 0x1c	reserved
SEL_{EVT} + 0x0d	#GP	SEL_{EVT} + 0x1d	reserved
SEL_{EVT} + 0x0e	#PF	SEL_{EVT} + 0x1e	reserved
SEL_{EVT} + 0x0f	reserved	SEL_{EVT} + 0x1f	reserved

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

Guest Intercepts (VMX)

SEL_{OBJ}	Intercept
SEL_{EVT} + 0x00	Exception or NMI*
SEL_{EVT} + 0x01	External Interrupt*
SEL_{EVT} + 0x02	Triple Fault†
SEL_{EVT} + 0x03	INIT†
SEL_{EVT} + 0x04	SIPI†
SEL_{EVT} + 0x05	I/O SMI
SEL_{EVT} + 0x06	Other SMI
SEL_{EVT} + 0x07	Interrupt Window
SEL_{EVT} + 0x08	NMI Window
SEL_{EVT} + 0x09	Task Switch†
SEL_{EVT} + 0x0a	CPUID†
SEL_{EVT} + 0x0b	GETSEC†
SEL_{EVT} + 0x0c	HLT†
SEL_{EVT} + 0x0d	INVD†
SEL_{EVT} + 0x0e	INVLPG
SEL_{EVT} + 0x0f	RDPMC
SEL_{EVT} + 0x10	RDTSC
SEL_{EVT} + 0x11	RSM
SEL_{EVT} + 0x12	VMCALL
SEL_{EVT} + 0x13	VMCLEAR
SEL_{EVT} + 0x14	VMLAUNCH
SEL_{EVT} + 0x15	VMPTRLD
SEL_{EVT} + 0x16	VMPTRST
SEL_{EVT} + 0x17	VMREAD
SEL_{EVT} + 0x18	VMRESUME
SEL_{EVT} + 0x19	VMWRITE
SEL_{EVT} + 0x1a	VMXOFF
SEL_{EVT} + 0x1b	VMXON
SEL_{EVT} + 0x1c	CR Access*
SEL_{EVT} + 0x1d	DR Access
SEL_{EVT} + 0x1e	I/O Access†
SEL_{EVT} + 0x1f	RDMSR†
SEL_{EVT} + 0x20	WRMSR†
SEL_{EVT} + 0x21	VM Entry Failure (State)†
SEL_{EVT} + 0x22	VM Entry Failure (MSR)
SEL_{EVT} + 0x23	reserved
SEL_{EVT} + 0x24	MWAIT
SEL_{EVT} + 0x25	MTF
SEL_{EVT} + 0x26	reserved
SEL_{EVT} + 0x27	MONITOR

SEL_{OBJ}	Intercept
SEL_{EVT} + 0x28	PAUSE
SEL_{EVT} + 0x29	VM Entry Failure (MCE)
SEL_{EVT} + 0x2a	reserved
SEL_{EVT} + 0x2b	TPR Below Threshold
SEL_{EVT} + 0x2c	APIC Access
SEL_{EVT} + 0x2d	Virtualized EOI
SEL_{EVT} + 0x2e	GDTR/IDTR Access
SEL_{EVT} + 0x2f	LDTR/TR Access
SEL_{EVT} + 0x30	EPT Violation†
SEL_{EVT} + 0x31	EPT Misconfiguration
SEL_{EVT} + 0x32	INVEPT
SEL_{EVT} + 0x33	RDTSCP
SEL_{EVT} + 0x34	Preemption Timer
SEL_{EVT} + 0x35	INVVPID
SEL_{EVT} + 0x36	WBINVD, WBNOINVD
SEL_{EVT} + 0x37	XSETBV
SEL_{EVT} + 0x38	APIC Write
SEL_{EVT} + 0x39	RDRAND
SEL_{EVT} + 0x3a	INVPCID
SEL_{EVT} + 0x3b	VMFUNC
SEL_{EVT} + 0x3c	ENCLS
SEL_{EVT} + 0x3d	RDSEED
SEL_{EVT} + 0x3e	PML Log Full
SEL_{EVT} + 0x3f	XSAVES
SEL_{EVT} + 0x40	XRSTORS
SEL_{EVT} + 0x41	reserved
SEL_{EVT} + 0x42	SPP Miss / Misconfiguration
SEL_{EVT} + 0x43	UMWAIT
SEL_{EVT} + 0x44	TPAUSE
SEL_{EVT} + 0x45	LOADIWKEY
SEL_{EVT} + 0x46	reserved
SEL_{EVT} + 0x47	reserved
SEL_{EVT} + 0x48	ENQCMD PASID Failure
SEL_{EVT} + 0x49	ENQCMDS PASID Failure
SEL_{EVT} + 0x4a	Bus Lock
SEL_{EVT} + 0x4b	Notify Window
SEL_{EVT} + 0x4c	SEAMCALL
SEL_{EVT} + 0x4d	TDCALL
SEL_{EVT} + 0x4e	reserved
SEL_{EVT} + 0x4f	reserved

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

8.6.2 Microhypervisor Events

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

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

- SEL_{ARCH} = $SEL_{HST/ARCH}$ for events that occurred in the host.
- SEL_{ARCH} = $SEL_{GST/ARCH}$ 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

48	32	16	0	48	32	16	0	
SEL_MSR				SEL_PIO				+0x260
SEL_GST				IA32_TSC_AUX				+0x250
IA32_KERNEL_GS_BASE				IA32_FMASK				+0x240
IA32_LSTAR				IA32_STAR				+0x230
IA32_EFER				IA32_PAT				+0x220
IA32_SYSENTER_EIP				IA32_SYSENTER_ESP				+0x210
IA32_SYSENTER_CS				IA32_APIC_BASE				+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

*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

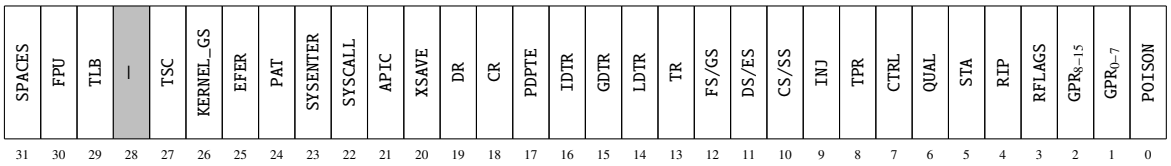
Field	Description
U	0 = Segment Usable 1 = Segment Unusable
G	Granularity
D/B	0 = 16-bit segment 1 = 32-bit segment
L	64-bit mode active (CS only)
AVL	Available for use by system software
P	Segment Present
DPL	Descriptor Privilege Level
S	0 = System 1 = Code or Data
Type	Segment Type

8.7.2.2 Encoding: Interruption Information

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:



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 Execution Context State	Guest Execution Context State
POISON	w	Kills the EC_{HST}	Kills the EC_{GST}
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
APIC	w	-	IA32_APIC_BASE§
SYSCALL	rw	-	IA32_{STAR,LSTAR,FMASK}
SYSENTER	rw	-	IA32_SYSENTER_{CS,ESP,EIP}
PAT	rw	-	IA32_PAT
EFER	rw	-	IA32_EFER
KERNEL_GS	rw	-	IA32_KERNEL_GS_BASE
TSC	rw	-	IA32_TSC_AUX
TLB	w	-	Invalidates the TLB for the vCPU
SPACES	w	-	SEL_GST, SEL_PIO, SEL_MSR

*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).
§APIC access page guest-physical address in the vCPU's currently assigned guest space.

8.7.4 Memory Attribute Descriptor

The Memory Attribute Descriptor (MAD) describes page attributes for a CAP_{MEM} delegation from $SPC_{HST_{NOVA}}$.

The fields are defined as follows:

CA

Specifies the cacheability attributes of the page according to the following table:

Encoding	Cacheability	Description
0x0	WB	Write Back
0x1	WT	Write Through
0x2	WC	Write Combining
0x3	UC	Strong Uncacheable
0x4	WP	Write Protected
0x5	-	*reserved*
0x6	-	*reserved*
0x7	-	*reserved*

KI

Specifies the key identifier of the cryptographic key to be used for memory encryption – must be ≤ KI_{MAX}.

Please refer to [4, 5] for the architectural behavior of these attributes.

8.7.5 Device Assignment Descriptor

The Device Assignment Descriptor (DAD) describes a device to be assigned to a DMA Space (SPC_{DMA}).

On x86, IOMMU resources need not be specified.

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. A Host 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.

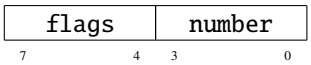


Figure 8.1: Hypercall Identifier

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

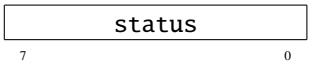


Figure 8.2: Status Code

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

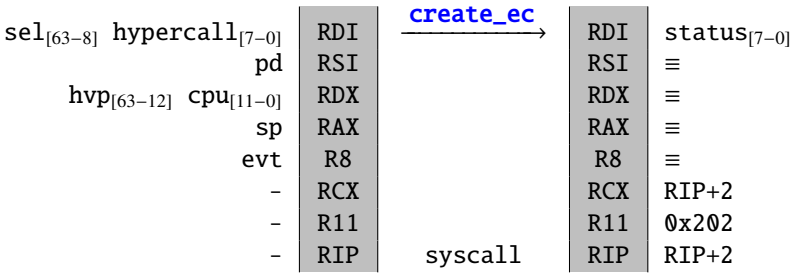
IPC Reply

	$hypercall_{[7-0]}$	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

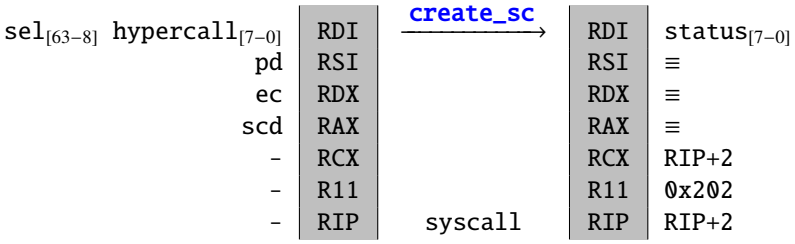
Create Protection Domain

$sel_{[63-8]}$	$hypercall_{[7-0]}$	RDI	create_pd →		RDI	$status_{[7-0]}$
	pd	RSI			RSI	≡
	-	RCX			RCX	RIP+2
	-	R11			R11	0x202
	-	RIP		syscall	RIP	RIP+2

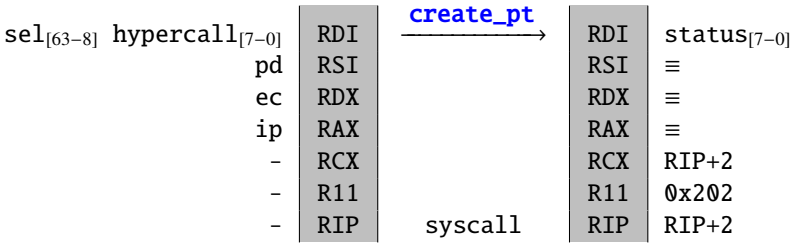
Create Execution Context



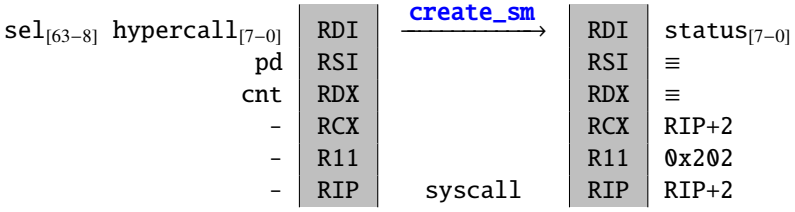
Create Scheduling Context



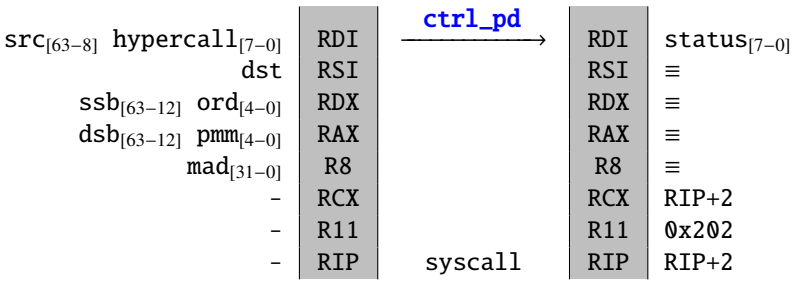
Create Portal



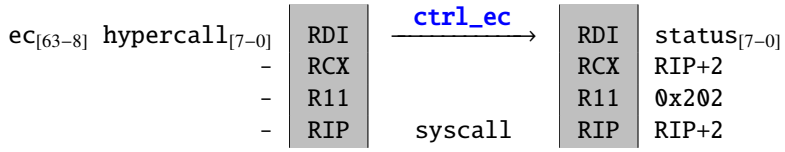
Create Semaphore



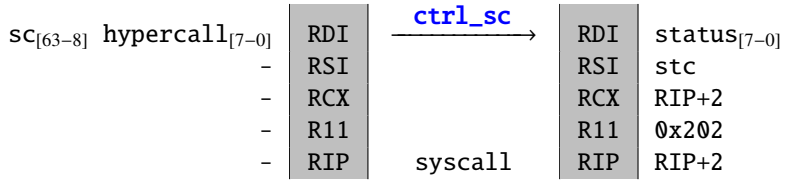
Control Protection Domain



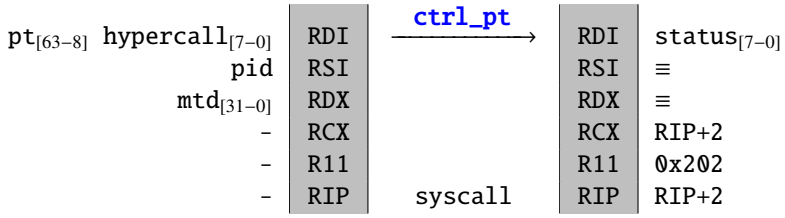
Control Execution Context



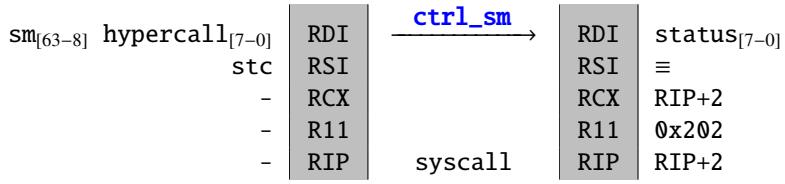
Control Scheduling Context



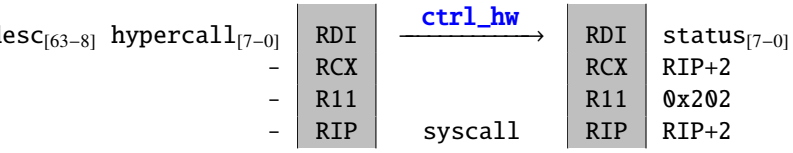
Control Portal



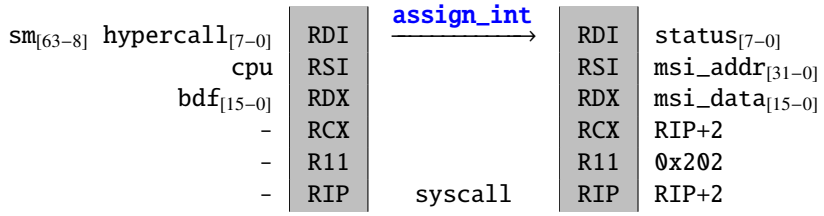
Control Semaphore



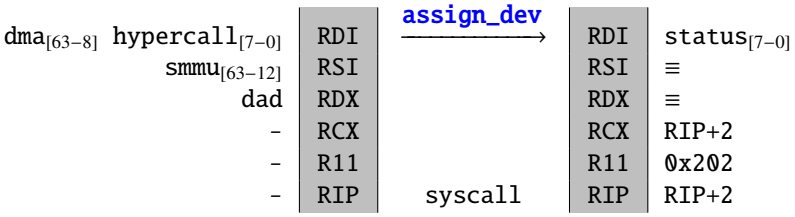
Control Hardware



Assign Interrupt



Assign Device



Part V

Appendix

A Acronyms

ACPI	Advanced Configuration and Power Interface [6]
BDF	Bus/Device/Function [19, 20]
BSP	Bootstrap Processor
CAP	Capability
CAP_0	Null Capability
CAP_{OBJ}	Object Capability
$CAP_{OBJ_{OBJ}}$	Object Space Capability
$CAP_{OBJ_{HST}}$	Host Space Capability
$CAP_{OBJ_{GST}}$	Guest Space Capability
$CAP_{OBJ_{DMA}}$	DMA Space Capability
$CAP_{OBJ_{PIO}}$	PIO Space Capability
$CAP_{OBJ_{MSR}}$	MSR Space Capability
$CAP_{OBJ_{PD}}$	PD Capability
$CAP_{OBJ_{EC}}$	EC Capability
$CAP_{OBJ_{SC}}$	SC Capability
$CAP_{OBJ_{PT}}$	PT Capability
$CAP_{OBJ_{SM}}$	SM Capability
CAP_{MEM}	Memory Capability
CAP_{PIO}	PIO Capability
CAP_{MSR}	MSR 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
DVA	DMA-Virtual Address
EC	Execution Context
EC_{HST}	Host Execution Context
EC_{GST}	Guest Execution Context
$EC_{CURRENT}$	Current Execution Context
EC_{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
GPA	Guest-Physical Address [Arm, x86]
HIP	Hypervisor Information Page [Arm, x86]
HVA	Host-Virtual Address [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
MAD	Memory Attribute Descriptor [Arm, x86]
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	**N**OVA **O**S **V**irtualization **A**rchitecture [2]
PCI	Peripheral Component Interconnect [19, 20]
PD	Protection Domain
$\mathbf{PD_{ROOT}}$	Root Protection Domain
PID	Portal Identifier
PT	Portal
QOS	Quality Of Service
SC	Scheduling Context
$\mathbf{SC_{CURRENT}}$	Current Scheduling Context
$\mathbf{SC_{ROOT}}$	Root Scheduling Context
SCD	Scheduling Context Descriptor
SEL	Capability Selector
$\mathbf{SEL_{EVT}}$	Event Selector Base [Arm, x86]
$\mathbf{SEL_{OBJ}}$	Object Capability Selector
$\mathbf{SEL_{HST}}$	Host Capability Selector
$\mathbf{SEL_{GST}}$	Guest Capability Selector
$\mathbf{SEL_{DMA}}$	DMA Capability Selector
$\mathbf{SEL_{PIO}}$	PIO Capability Selector
$\mathbf{SEL_{MSR}}$	MSR 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$_{OBJ}$	Object Space
SPC$_{HST}$	Host Space
SPC$_{GST}$	Guest Space
SPC$_{DMA}$	DMA Space
SPC$_{PIO}$	PIO Space
SPC$_{MSR}$	MSR Space
SPC$_{OBJ_{CURRENT}}$	Current Object Space
SPC$_{OBJ_{NOVA}}$	NOVA Object Space
SPC$_{HST_{NOVA}}$	NOVA Host Space
SPC$_{PIO_{NOVA}}$	NOVA PIO Space
SPC$_{MSR_{NOVA}}$	NOVA MSR Space
SPC$_{OBJ_{ROOT}}$	Root Object Space
SPC$_{HST_{ROOT}}$	Root Host Space
SPC$_{PIO_{ROOT}}$	Root PIO Space
STC	System Time Counter
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. v

[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, 75

[3] *Arm Architecture Reference Manual ARMv8, for ARMv8-A Architecture Profile*. Arm Limited, 2022. URL https://developer.arm.com/documentation/ddi0487/. Document Number: DDI0487. 8, 49, 54, 55, 74, 75

[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. 8, 64, 68, 69, 74, 75

[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. 8, 68, 69, 74, 75

[6] *Advanced Configuration and Power Interface (ACPI) Specification*. UEFI Forum, Inc., 2022. URL https://uefi.org/specifications. Version 6.5. 43, 74

[7] *Unified Extensible Firmware Interface (UEFI) Specification*. UEFI Forum, Inc., 2022. URL https://uefi.org/specifications. Version 2.10. 43, 61, 76

[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. 46, 59, 61

[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. 46, 59, 61

[10] *Arm Generic Interrupt Controller Architecture Specification Version 2*. Arm Limited, 2013. URL https://developer.arm.com/documentation/ihi0048/. Document Number: IHI0048. 48, 74

[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. 48, 74

[12] *Arm System Memory Management Unit Architecture Specification Version 2*. Arm Limited, 2016. URL https://developer.arm.com/documentation/ihi0062/. Document Number: IHI0062. 48, 74, 75

[13] *Arm System Memory Management Unit Architecture Specification Version 3*. Arm Limited, 2021. URL https://developer.arm.com/documentation/ihi0070/. Document Number: IHI0070. 48, 74, 75

[14] *Arm SMC Calling Convention*. Arm Limited, 2022. URL https://developer.arm.com/documentation/den0028/. Document Number: DEN0028. 58

[15] *Arm True Random Number Generator Firmware Interface*. Arm Limited, 2022. URL https://developer.arm.com/documentation/den0098/. Document Number: DEN0098. 58

[16] *Arm PCI Configuration Space Access Firmware Interface*. Arm Limited, 2022. URL https://developer.arm.com/documentation/den0115/. Document Number: DEN0115. 58

[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. 60, 75

[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. 60, 75

[19] *PCI Local Bus Specification*. PCI-SIG, 2004. URL https://pcisig.com/specifications. Revision 3.0. 74, 75

[20] *PCI Express Base Specification*. PCI-SIG, 2019. URL https://pcisig.com/specifications. Revision 5.0. 74, 75

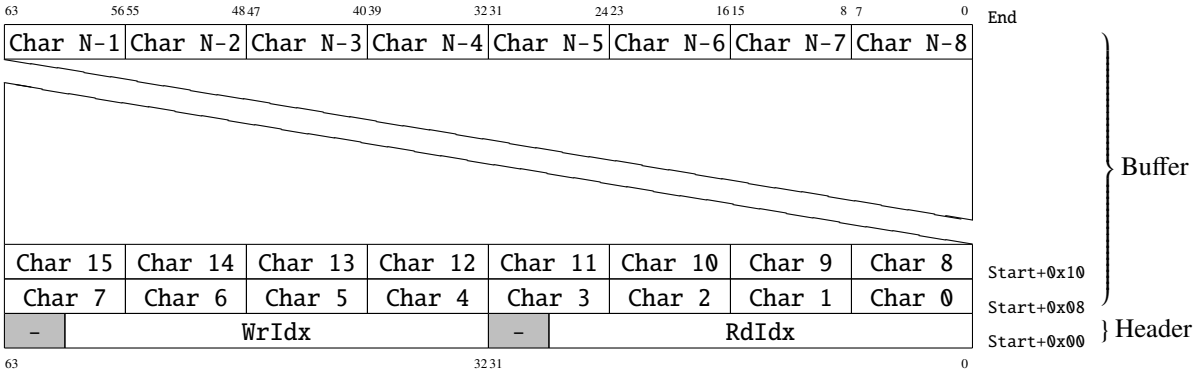
[21] *Executable and Linking Format (ELF) Specification*. TIS Committee, 1995. URL https://refspecs.linuxbase.org/elf/elf.pdf. Version 1.2. 74

[22] *Devicetree Specification*. Linaro Limited, 2020. URL https://www.devicetree.org/specifications. Version 0.3. 74

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