NOVA Microhypervisor Interface Specification

Udo Steinberg
udo@hypervisor.org

March 6, 2020

Contents

Notation

Throughout this document, the following symbols are used:

- ~ Indicates that the value of this parameter or field is **undefined**. Future versions of this specification may define a meaning for the parameter or field.
- _ Indicates that the value of this parameter or field is **ignored**. Future versions of this specification may define a meaning for the parameter or field.
- ≡ Indicates that the value of this parameter or field is **unchanged**. The microhypervisor will preserve the value across hypercalls.

Part I

Introduction

1 System Architecture

The **N**OVA **O**S **V**irtualization **A**rchitecture facilitates the coexistence of multiple legacy guest operating systems and a multi-server user-mode framework on a single platform [2]. The core system leverages virtualization technology provided by modern x86 or ARM platforms and comprises the NOVA microhypervisor and one or more Virtual-Machine Monitors (VMMs).

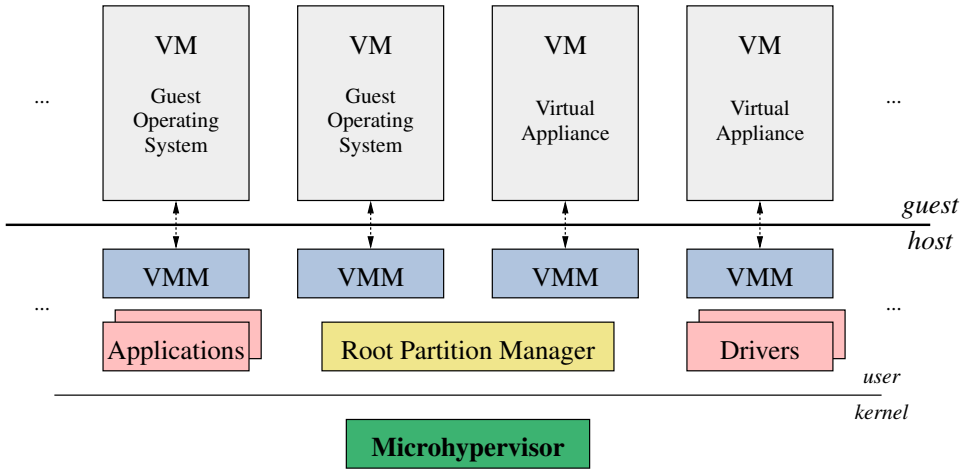


Figure 1.1: System Architecture

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

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

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

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

Part II

Basic Abstractions

2 Kernel Objects

2.1 Protection Domain

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

2.1.1 Object Space

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

2.1.2 Memory Space

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

2.1.3 I/O Port Space

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

2.2 Execution Context

1. The Execution Context (EC) is an abstraction for an activity within a PD.
2. Access to an Execution Context (EC) is controlled by an EC Object Capability ($\text{CAP}_{\text{OBJ}_{\text{EC}}}$).
3. An EC is permanently bound to the PD in which it was created.
4. An EC may optionally have an SC bound to it.
5. There exist two flavors of execution context:
   - Threads
   - Virtual CPUs
6. An EC comprises the following state:
   - Reference to PD (2.1)
   - Event Selector Base (SEL_{EVT}) (**??**)
   - User Thread Control Block (UTCB) (3.3)
   - CPU Number (CPU) registers (architecture dependent)
   - Floating Point Unit (FPU) registers (architecture dependent)

2.3 Scheduling Context

1. The Scheduling Context (SC) is a unit of dispatching and prioritization.
2. Access to a Scheduling Context (SC) is controlled by an SC Object Capability ($\texttt{CAP}_{\texttt{OBJ}_{\texttt{SC}}}$).
3. An SC is permanently bound to exactly one physical CPU.
4. At any point in time, an SC is bound to exactly one EC.
5. Donation of an SC to another EC temporarily binds the SC to that other EC.
6. A scheduling context comprises the following state:
   - Reference to EC (2.2)
   - Time quantum
   - Priority

2.4 Portal

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

2.5 Semaphore

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

Part III

Application Programming Interface

3 Data Types

3.1 Capability

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

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

3.1.1 Null Capability

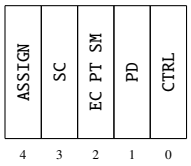
A Null Capability (CAP_0) does not refer to anything and carries no permissions.

3.1.2 Object Capability

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

3.1.2.1 PD Object Capability

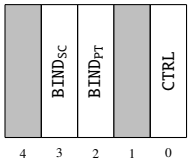
A PD Object Capability ($\text{CAP}_{\text{OBJ}_{\text{PD}}}$) refers to a Protection Domain (PD) and carries the following permissions:



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

3.1.2.2 EC Object Capability

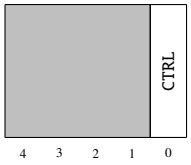
An EC Object Capability ($\text{CAP}_{\text{OBJ}_{\text{EC}}}$) refers to an Execution Context (EC) and carries the following permissions:



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

3.1.2.3 SC Object Capability

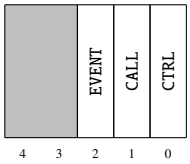
An SC Object Capability ($\text{CAP}_{\text{OBJ}_{\text{SC}}}$) refers to a Scheduling Context (SC) and carries the following permissions:



CTRL	ctrl_sc permitted if set.

3.1.2.4 PT Object Capability

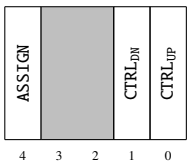
A PT Object Capability ($\texttt{CAP}_{\texttt{OBJ}_{\texttt{PT}}}$) refers to a Portal (PT) and carries the following permissions:



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

3.1.2.5 SM Object Capability

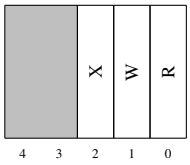
An SM Object Capability ($\texttt{CAP}_{\texttt{OBJ}_{\texttt{SM}}}$) refers to a Semaphore (SM) and carries the following permissions:



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

3.1.3 Memory Capability

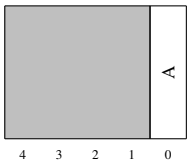
A Memory Capability ($\texttt{CAP}_{\texttt{MEM}}$) is stored in the Memory Space ($\texttt{SPC}_{\texttt{MEM}}$) of a PD, refers to a 4KB page frame, and carries the following permissions:



R	the memory page is readable if set.
W	the memory page is writable if set.
X	the memory page is executable if set.

3.1.4 I/O Port Capability

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



A	the I/O port is accessible if set.

3.2 Capability Selector

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

- An Object Capability Selector ($\texttt{SEL}_{\texttt{OBJ}}$) serves as an index into the Object Space ($\texttt{SPC}_{\texttt{OBJ}}$) of a Protection Domain (PD) and selects a slot that either contains an Object Capability ($\texttt{CAP}_{\texttt{OBJ}}$) or a Null Capability ($\texttt{CAP}_{0}$).
- A Memory Capability Selector ($\texttt{SEL}_{\texttt{MEM}}$) serves as an index into the Memory Space ($\texttt{SPC}_{\texttt{MEM}}$) of a Protection Domain (PD) and selects a slot that either contains a Memory Capability ($\texttt{CAP}_{\texttt{MEM}}$) or a Null Capability ($\texttt{CAP}_{0}$).
- A I/O Port Capability Selector ($\texttt{SEL}_{\texttt{PIO}}$) serves as an index into the I/O Port Space ($\texttt{SPC}_{\texttt{PIO}}$) of a Protection Domain (PD) and selects a slot that either contains a I/O Port Capability ($\texttt{CAP}_{\texttt{PIO}}$) or a Null Capability ($\texttt{CAP}_{0}$).

3.3 User Thread Control Block

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

The UTCB always has a size of one page (4096 bytes) and is used as inbox/outbox during IPC as follows:

- `ipc_call` transfers a message from the UTCB of the caller EC to the UTCB of the callee EC.
- `ipc_reply` transfers a message from the UTCB of the callee EC to the UTCB of the caller EC.

Data Transfer

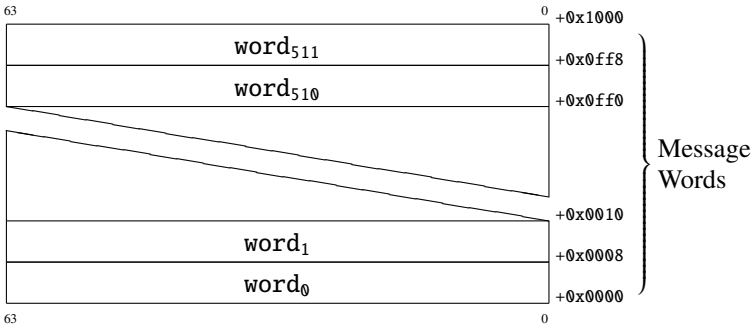
The data transfer from one UTCB to another UTCB is defined as follows:

- The microhypervisor copies the data using the CPU on which the caller/callee EC execute.
- The data is copied from low words to high words, beginning with $word_0$.
- The granularity of the loads and stores used for copying is **undefined**.
- Loads from and stores to the UTCB by the microhypervisor use **relaxed** memory ordering.

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

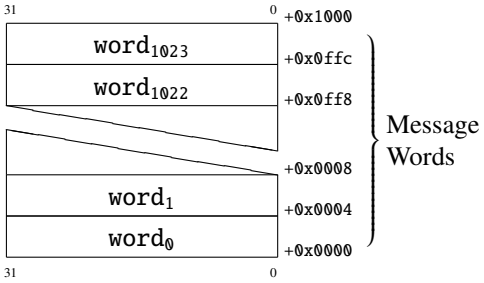
64-bit Architectures

A 64-bit UTCB consists of 512 message words. Each message word has a size of 8 bytes.



32-bit Architectures

A 32-bit UTCB consists of 1024 message words. Each message word has a size of 4 bytes.



4 Hypercalls

4.1 Definitions

4.1.1 Hypercall Numbers

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

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

4.1.2 Status Codes

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

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

4.1.3 Space Type

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

4.1.4 Table Type

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

4.1.5 Cacheability Attributes

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

4.1.6 Shareability Attributes

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

4.2 Communication

4.2.1 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) bound to the specified Portal (PT).

Prior to the hypercall:

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

If the hypercall completed successfully:

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

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

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

BAD_CPU

- Caller EC and callee EC are on different CPUs.

TIMEOUT

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

ABORTED

- The callee EC is dead and the operation aborted.

4.2.2 IPC Reply

Parameters:

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

Flags:

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

4.3 Object Creation

4.3.1 Create Protection Domain

Parameters:

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

Flags:

0	0	0	0
3	2	1	0

Description:

Creates a new Protection Domain (PD).

Prior to the hypercall:

- { $PD_{CURRENT}$, SEL_{OBJ} own } must refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) with permission PD.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } must refer to a Null Capability (CAP_{0}).

If the hypercall completed successfully:

- A new Protection Domain (PD) has been created.
- The resources for the created PD are accounted to the PD referred to by { $PD_{CURRENT}$, SEL_{OBJ} own }.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } refers to a PD Object Capability ($CAP_{OBJ_{PD}}$) for the created PD.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

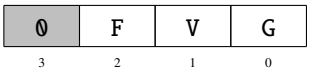
- { $PD_{CURRENT}$, SEL_{OBJ} own } did not refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) or that capability had insufficient permissions.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } did not refer to a Null Capability (CAP_{0}).

4.3.2 Create Execution Context

Parameters:

```
status = create_ec (SEL_OBJ   sel,        // Created EC
                    SEL_OBJ   own,        // Owner PD
                    SEL_MEM   utcb,       // UTCB 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:

- { $PD_{CURRENT}$, SEL_{OBJ} own } must refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) with permission EC.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } must refer to a Null Capability (CAP_0).

If the hypercall completed successfully:

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

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $PD_{CURRENT}$, SEL_{OBJ} own } did not refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) or that capability had insufficient permissions.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } did not refer to a Null Capability (CAP_0).

BAD_CPU

- The CPU number is invalid.

BAD_FTR

- Virtual CPUs are not supported on the machine.

BAD_PAR

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

4.3.3 Create Scheduling Context

Parameters:

```
status = create_sc (SEL_OBJ  sel,          // Created SC
                    SEL_OBJ  own,          // Owner PD
                    SEL_OBJ  ec,           // Bound EC
                    QPD      qpd);         // Scheduling Parameters
```

Flags:

Description:

Creates a new Scheduling Context (SC).

Prior to the hypercall:

- { $PD_{CURRENT}$, SEL_{OBJ} own } must refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) with permission SC.
- { $PD_{CURRENT}$, SEL_{OBJ} ec } must refer to an EC Object Capability ($CAP_{OBJ_{EC}}$) with permission $BIND_{SC}$.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } must refer to a Null Capability (CAP_{0}).

If the hypercall completed successfully:

- A new Scheduling Context (SC) has been created.
- The created SC is bound to the EC referred to by { $PD_{CURRENT}$, SEL_{OBJ} ec } on the CPU of that EC with its scheduling parameters set to qpd.
- The resources for the created SC are accounted to the PD referred to by { $PD_{CURRENT}$, SEL_{OBJ} own }.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } refers to an SC Object Capability ($CAP_{OBJ_{SC}}$) for the created SC.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $PD_{CURRENT}$, SEL_{OBJ} own } did not refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) or that capability had insufficient permissions.
- { $PD_{CURRENT}$, SEL_{OBJ} ec } did not refer to a EC Object Capability ($CAP_{OBJ_{EC}}$) or that capability had insufficient permissions.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } did not refer to a Null Capability (CAP_{0}).
- Binding the SC to the EC failed, e.g. because the EC is a local EC.

BAD_PAR

- qpd time quantum or priority is zero.

4.3.4 Create Portal

Parameters:

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

Flags:

Description:

Creates a new Portal (PT).

Prior to the hypercall:

- { $PD_{CURRENT}$, SEL_{OBJ} own } must refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) with permission PT.
- { $PD_{CURRENT}$, SEL_{OBJ} ec } must refer to an EC Object Capability ($CAP_{OBJ_{EC}}$) with permission $BIND_{PT}$.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } must refer to a Null Capability (CAP_0).

If the hypercall completed successfully:

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

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

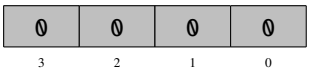
- { $PD_{CURRENT}$, SEL_{OBJ} own } did not refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) or that capability had insufficient permissions.
- { $PD_{CURRENT}$, SEL_{OBJ} ec } did not refer to a EC Object Capability ($CAP_{OBJ_{EC}}$) or that capability had insufficient permissions.
- { $PD_{CURRENT}$,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

4.3.5 Create Semaphore

Parameters:

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

Flags:



Description:

Creates a new Semaphore (SM).

Prior to the hypercall:

- { $PD_{CURRENT}$, SEL_{OBJ} own } must refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) with permission SM.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } must refer to a Null Capability (CAP_{0}).

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 are accounted to the PD referred to by { $PD_{CURRENT}$, SEL_{OBJ} own }.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } refers to an SM Object Capability ($CAP_{OBJ_{SM}}$) for the created SM.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { $PD_{CURRENT}$, SEL_{OBJ} own } did not refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) or that capability had insufficient permissions.
- { $PD_{CURRENT}$, SEL_{OBJ} sel } did not refer to a Null Capability (CAP_{0}).

4.4 Object Control

4.4.1 Control Protection Domain

Parameters:

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

Flags:

Description:

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

Prior to the hypercall:

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

If the hypercall completed successfully:

- If **spc=SPC_{OBJ}**: All CAP_{OBJ} and CAP_{0} from source SEL range { PD spd, SEL_{OBJ} src...src+2^{ord}-1 } were delegated to destination SEL range { PD dpd, SEL_{OBJ} dst...dst+2^{ord}-1 }. Any pre-existing CAP_{OBJ} in the destination selector range were revoked. The parameters tbl, ca and sh were ignored.
- If **spc=SPC_{MEM}**: All CAP_{MEM} and CAP_{0} from source SEL range { PD spd, SEL_{MEM} src...src+2^{ord}-1 } were delegated to destination SEL range { PD dpd, SEL_{MEM} dst...dst+2^{ord}-1 }. Any pre-existing CAP_{MEM} in the destination selector range were revoked.
- If **spc=SPC_{PIO}**: All CAP_{PIO} and CAP_{0} from source SEL range { PD spd, SEL_{PIO} src...src+2^{ord}-1 } were delegated to destination SEL range { PD dpd, SEL_{PIO} dst...dst+2^{ord}-1 }. Any pre-existing CAP_{PIO} in the destination selector range were revoked. The parameters tbl, ca and sh were ignored.
- 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 would have had zero permissions after masking, then the destination capability is now a Null Capability (CAP_{0}).

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

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

BAD_PAR

- SEL `src` or SEL `dst` were not order-aligned.
- TYPE$_{\texttt{SPC}}$ `spc` or TYPE$_{\texttt{TBL}}$ `tbl` were not valid, i.e. not supported by the architecture.
- ATTR$_{\texttt{CA}}$ `ca` or ATTR$_{\texttt{SH}}$ `sh` were not valid, i.e. not supported by the architecture.

4.4.2 Control Execution Context

Parameters:

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

Flags:

Description:

Prior to the hypercall:

- { PD_CURRENT, SEL_OBJ ec } must refer to a EC Object Capability (CAP_OBJ_EC) with permission CTRL.

If the hypercall completed successfully:

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

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

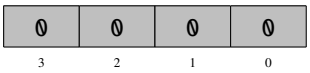
- { PD_CURRENT, SEL_OBJ ec } did not refer to a EC Object Capability (CAP_OBJ_EC) or that capability had insufficient permissions.

4.4.3 Control Scheduling Context

Parameters:

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

Flags:



Description:

Prior to the hypercall:

- { PD_CURRENT, SEL_OBJ sc } must refer to an SC Object Capability (CAP_OBJ_SC) with permission CTRL.

If the hypercall completed successfully:

- The microhypervisor has returned the total consumed execution time in `ticks` for the SC referred to by { PD_CURRENT, SEL_OBJ sc }.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

- { PD_CURRENT, SEL_OBJ sc } did not refer to an SC Object Capability (CAP_OBJ_SC) or that capability had insufficient permissions.

4.4.4 Control Portal

Parameters:

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

Flags:

Description:

Prior to the hypercall:

- { $PD_{CURRENT}$, SEL_{OBJ} pt } must refer to a PT Object Capability ($CAP_{OBJ_{PT}}$) 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 { $PD_{CURRENT}$, SEL_{OBJ} pt }.
- Subsequent portal traversals will use the new MTD and return the new PID.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CAP

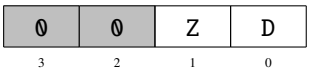
- { $PD_{CURRENT}$, SEL_{OBJ} pt } did not refer to a PT Object Capability ($CAP_{OBJ_{PT}}$) or that capability had insufficient permissions.

4.4.5 Control Semaphore

Parameters:

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

Flags:



Description:

Prior to the hypercall:

- If **D=0 (Up)**: { $PD_{CURRENT}$, SEL_{OBJ} sm } must refer to a SM Object Capability ($CAP_{OBJ_{SM}}$) with permission $CTRL_{UP}$.
- If **D=1 (Down)**: { $PD_{CURRENT}$, SEL_{OBJ} sm } must refer to a SM Object Capability ($CAP_{OBJ_{SM}}$) with permission $CTRL_{DN}$.

If the hypercall completed successfully:

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

Status:

SUCCESS

- The hypercall completed successfully.

TIMEOUT

- Hypercall aborted due to timeout.

OVRFLOW

- Hypercall aborted due to semaphore counter overflow.

BAD_CAP

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

4.4.6 Control Hardware

Parameters:

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

Flags:

3	2	1	0
1	1	1	1

Description:

Performs a firmware call via SMC.

Prior to the hypercall:

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

If the hypercall completed successfully:

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

Status:

SUCCESS

- The hypercall completed successfully.

BAD_HYP

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

BAD_PAR

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

BAD_FTR

- The CPU does not support SMCs.

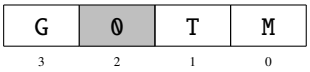
4.5 Interrupt and Device Assignment

4.5.1 Assign Interrupt

Parameters:

```
status = assign_int (SEL_OBJ sm,              // Interrupt Semaphore
                     UINT    cpu);            // CPU Number
```

Flags:



Description:

Configures a platform interrupt and routes it to the specified CPU.

Prior to the hypercall:

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

If the hypercall completed successfully:

- The platform interrupt referred to by { $PD_{CURRENT}$, SEL_{OBJ} sm } has been routed to the CPU cpu.
- If **M=0 (Unmask)**: The interrupt is now unmasked, i.e. it will be signaled on the semaphore.
- If **M=1 (Mask)**: The interrupt is now masked, i.e. it will not be signaled on the semaphore.
- If **T=0 (Level)**: The interrupt is now configured for level-triggered operation.
- If **T=1 (Edge)**: The interrupt is now configured for edge-triggered operation.
- If **G=0 (Host)**: The interrupt is now host-owned.
- If **G=1 (Guest)**: The interrupt is now guest-owned (VM pass-through).

Status:

SUCCESS

- The hypercall completed successfully.

BAD_CPU

- The specified CPU number was invalid.

BAD_CAP

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

4.5.2 Assign Device

Parameters:

```
status = assign_dev (SEL_OBJ pd,            // Protection Domain
                     UINT    ctx,           // Translation Context
                     UINT    smg,           // Stream Mapping Group
                     UINT    sid,           // Stream Identifier
                     TYPE_TBL tbl);         // Table Type
```

Flags:

Description:

Assigns the specified device/stream to the specified Protection Domain (PD).

Prior to the hypercall:

- $PD_{CURRENT}$ must be the Root Protection Domain (PD_{ROOT}).
- { $PD_{CURRENT}$, SEL_{OBJ} pd } must refer to a PD Object Capability ($CAP_{OBJ_{PD}}$) with permission ASSIGN.
- ctx must be within the number of translation contexts supported by the hardware (see 5.3).
- smg must be within the number of stream mapping groups supported by the hardware (see 5.3).
- sid must be within the number of stream identifiers supported by the hardware.
- $TYPE_{TBL}$ tbl must refer to a DMA page table.

If the hypercall completed successfully:

- The device/stream, identified by stream identifier sid, has been assigned to the Protection Domain (PD) referred to by { $PD_{CURRENT}$, SEL_{OBJ} pd }.
- DMA transactions issued by that device/stream will be managed using stream mapping group smg and translation context ctx. Prior users of stream mapping group smg or translation context ctx have been unconfigured.
- DMA transactions issued by that device/stream will be translated by the DMA page table referred to by $TYPE_{TBL}$ tbl of the assigned PD.

Status:

SUCCESS

- The hypercall completed successfully.

BAD_HYP

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

BAD_CAP

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

BAD_PAR

- At least one of the parameters ctx, smg, sid or tbl was not valid.

5 Booting

5.1 Microhypervisor

5.1.1 ELF Image Loading

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

```
readelf -l aarch64-qemu-hypervisor

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
```

5.1.2 ELF Image Launching

After loading the image into physical memory, the bootloader must invoke the NOVA microhypervisor by jumping to the physical address of the entry point of the NOVA microhypervisor ELF image with the following preconditions:

- Paging (MMU) must be disabled.
- I-Cache must be disabled.
- D-Cache must be disabled.
- The address range corresponding to the microhypervisor image must be clean to the Point of Coherence.

5.1.3 Special Resource Access

Possession of a PD Object Capability ($CAP_{OBJ_{PD}}$) for PD_{NOVA} allows the caller to invoke the ctrl_pd hypercall to take resources from the NOVA Protection Domain and grant them to another Protection Domain. In addition to memory regions not claimed by the NOVA microhypervisor, the following capabilities can be taken:

Interrupt Semaphores

{ PD_{NOVA}, SEL_{OBJ} 1024...1024+INT_{NUM} } refer to $CAP_{OBJ_{SM}}$ for interrupt semaphores, where INT_{NUM} is the maximum number of supported interrupts, as indicated by the HIP. These capabilities can be used with the assign_int and ctrl_sm hypercalls.

Console Signaling Semaphore

{ PD_{NOVA}, SEL_{OBJ} SEL_{NUM}-1 } refers to a $CAP_{OBJ_{SM}}$ for the signaling semaphore of the NOVA memory-buffer console. This capability can be used with the ctrl_sm hypercall.

5.2 Root Protection Domain

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

- $\texttt{PD}_{\texttt{ROOT}}$ – the Root Protection Domain
- $\texttt{EC}_{\texttt{ROOT}}$ – the Root Execution Context (executing in $\texttt{PD}_{\texttt{ROOT}}$)
- $\texttt{SC}_{\texttt{ROOT}}$ – the Root Scheduling Context (bound to $\texttt{EC}_{\texttt{ROOT}}$)

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

5.2.1 Initial Configuration

Prior to invoking the entry point of the Root Protection Domain ($\texttt{PD}_{\texttt{ROOT}}$) ELF image, using the Root Execution Context ($\texttt{EC}_{\texttt{ROOT}}$), the NOVA microhypervisor sets up $\texttt{PD}_{\texttt{ROOT}}$ as follows.

5.2.1.1 Object Space

The object space contains the following initial capabilities:

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

All other { $\texttt{PD}_{\texttt{ROOT}}$, $\texttt{SEL}_{\texttt{OBJ}}$ } refer to a Null Capability ($\texttt{CAP}_{\texttt{0}}$).

The value of $\texttt{SEL}_{\texttt{NUM}}$ is conveyed in the Hypervisor Information Page (HIP).

5.2.1.2 Memory Space

ELF Program Segments

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

Hypervisor Information Page

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

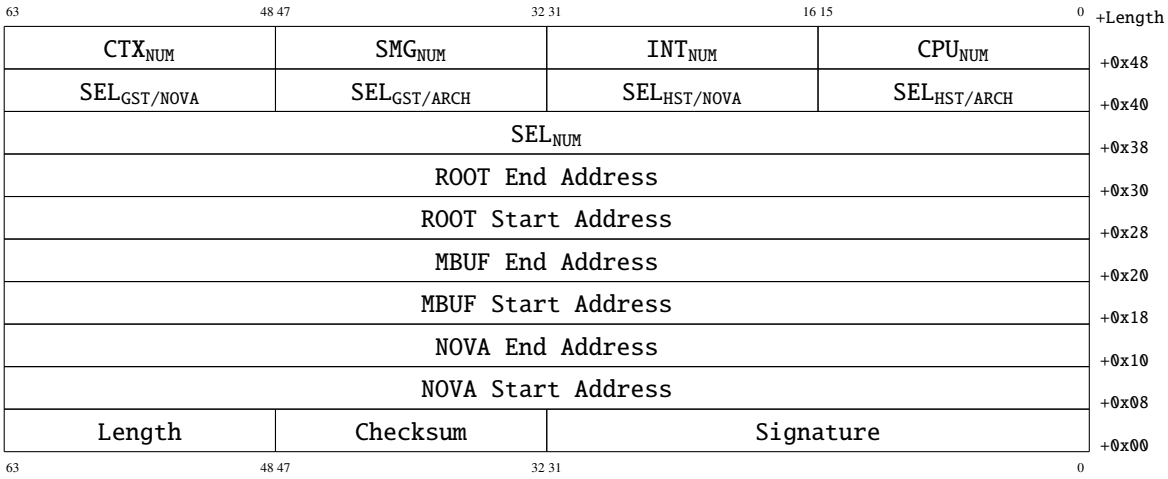
UTCB

The microhypervisor maps the User Thread Control Block of $\texttt{EC}_{\texttt{ROOT}}$ into the memory space 4KB below the address of the HIP.

All other { $\texttt{PD}_{\texttt{ROOT}}$, $\texttt{SEL}_{\texttt{MEM}}$ } refer to a Null Capability ($\texttt{CAP}_{\texttt{0}}$).

5.3 Hypervisor Information Page

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



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

Start and end address of the NOVA microhypervisor image in physical memory.

MBUF Start/End Address

Start and end address of the memory buffer console (see C.1) region in physical memory.

ROOT Start/End Address

Start and end address of the root protection domain image in physical memory.

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)

$SEL_{HST/NOVA}$

Number of additional capability selectors required for handling NOVA host events. (ARM)

$SEL_{GST/ARCH}$

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

$SEL_{GST/NOVA}$

Number of additional capability selectors required for handling NOVA guest events. (ARM)

CPU_{NUM}

Total number of CPUs that are online.

INT_{NUM}

Total number of interrupts that can be configured.

SMG_{NUM}

Total number of SMMU stream mapping groups that can be configured.

CTX_{NUM}

Total number of SMMU translation contexts that can be configured.

Part IV

Application Binary Interface

6 ABI aarch64

6.1 Virtual Memory

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

6.2 Initial State

Figure 6.1 details the state of the CPU registers when the microhypervisor has finished booting and transfers control to the Root Execution Context (EC_{ROOT}).

Register	Description	Note
`IP`	Virtual address of entry point from ELF header	
`SP`	Virtual address of hypervisor information page	
`X0`	Physical address of additional boot image	`0xffffffffffffffff` if not present
`X1`	Physical address of flattened device tree (FDT)	`0xffffffffffffffff` if not present
`Other`	~	

Figure 6.1: Initial State

6.3 Event-Specific Capability Selectors

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

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

6.3.1 Architectural Events

SEL_{OBJ}	Exception/Intercept
SEL_{EVT} + 0x0	Unknown Reason
SEL_{EVT} + 0x1	Trapped WFI or WFE
SEL_{EVT} + 0x2	reserved
SEL_{EVT} + 0x3	Trapped MCR or MRC
SEL_{EVT} + 0x4	Trapped MCRR or MRRC
SEL_{EVT} + 0x5	Trapped MCR or MRC
SEL_{EVT} + 0x6	Trapped LDC or STC
SEL_{EVT} + 0x7	SVE, SIMD, FPU
SEL_{EVT} + 0x8	Trapped VMRS Access
SEL_{EVT} + 0x9	Trapped PAuth Instruction
SEL_{EVT} + 0xa	reserved
SEL_{EVT} + 0xb	reserved
SEL_{EVT} + 0xc	Trapped MRRC
SEL_{EVT} + 0xd	reserved
SEL_{EVT} + 0xe	Illegal Execution State
SEL_{EVT} + 0xf	reserved
SEL_{EVT} + 0x10	reserved
SEL_{EVT} + 0x11	SVC (from AArch32 State)[1]
SEL_{EVT} + 0x12	HVC (from AArch32 State)
SEL_{EVT} + 0x13	SMC (from AArch32 State)
SEL_{EVT} + 0x14	reserved
SEL_{EVT} + 0x15	SVC (from AArch64 State)[1]
SEL_{EVT} + 0x16	HVC (from AArch64 State)
SEL_{EVT} + 0x17	SMC (from AArch64 State)
SEL_{EVT} + 0x18	Trapped MSR or MRS
SEL_{EVT} + 0x19	Trapped SVE
SEL_{EVT} + 0x1a	Trapped ERET
SEL_{EVT} + 0x1b	reserved
SEL_{EVT} + 0x1c	reserved
SEL_{EVT} + 0x1d	reserved
SEL_{EVT} + 0x1e	reserved
SEL_{EVT} + 0x1f	reserved

SEL_{OBJ}	Exception/Intercept
SEL_{EVT} + 0x20	Instruction Abort (lower EL)
SEL_{EVT} + 0x21	Instruction Abort (same EL)
SEL_{EVT} + 0x22	PC Alignment Fault
SEL_{EVT} + 0x23	reserved
SEL_{EVT} + 0x24	Data Abort (lower EL)
SEL_{EVT} + 0x25	Data Abort (same EL)
SEL_{EVT} + 0x26	SP Alignment Fault
SEL_{EVT} + 0x27	reserved
SEL_{EVT} + 0x28	Trapped FPU (AArch32)
SEL_{EVT} + 0x29	reserved
SEL_{EVT} + 0x2a	reserved
SEL_{EVT} + 0x2b	reserved
SEL_{EVT} + 0x2c	Trapped FPU (AArch64)
SEL_{EVT} + 0x2d	reserved
SEL_{EVT} + 0x2e	reserved
SEL_{EVT} + 0x2f	SError
SEL_{EVT} + 0x30	Breakpoint (lower EL)
SEL_{EVT} + 0x31	Breakpoint (same EL)
SEL_{EVT} + 0x32	Software Step (lower EL)
SEL_{EVT} + 0x33	Software Step (same EL)
SEL_{EVT} + 0x34	Watchpoint (lower EL)
SEL_{EVT} + 0x35	Watchpoint (same EL)
SEL_{EVT} + 0x36	reserved
SEL_{EVT} + 0x37	reserved
SEL_{EVT} + 0x38	BKPT (AArch32)
SEL_{EVT} + 0x39	reserved
SEL_{EVT} + 0x3a	Vector Catch (AArch32)
SEL_{EVT} + 0x3b	reserved
SEL_{EVT} + 0x3c	BRK (AArch64)
SEL_{EVT} + 0x3d	reserved
SEL_{EVT} + 0x3e	reserved
SEL_{EVT} + 0x3f	reserved

6.3.2 Microhypervisor Events

SEL_{OBJ}	Event
SEL_{EVT} + 0x40	Startup
SEL_{EVT} + 0x41	Recall
SEL_{EVT} + 0x42	Virtual Timer

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

6.4 User Thread Control Block: Architectural State

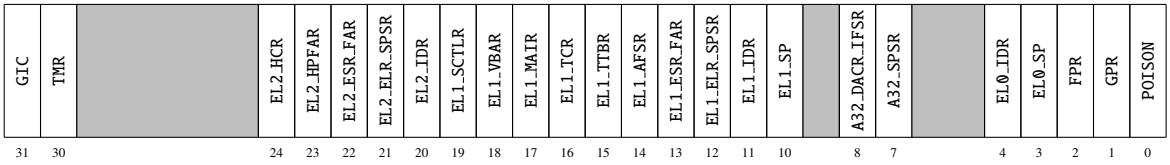
		VMCR	ELRSR	+0x290	GIC
LR15		LR14		+0x280	
LR13		LR12		+0x270	
LR11		LR10		+0x260	
LR9		LR8		+0x250	
LR7		LR6		+0x240	
LR5		LR4		+0x230	
LR3		LR2		+0x220	
LR1		LR0		+0x210	
CNTVOFF_EL2		CNTKCTL_EL1		+0x200	TMR
CNTV_CTL_EL0		CNTV_CVAL_EL0		+0x1f0	
HCR_EL2		HPFAR_EL2		+0x1e0	EL2
FAR_EL2		ESR_EL2		+0x1d0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

6.5 Message Transfer Descriptor: Regular IPC

The Message Transfer Descriptor (MTD), which controls the number of 64-bit message words transferred during regular IPC, as described in Section **??**, has the following layout:

6.6 Message Transfer Descriptor: Architectural State

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



Each MTD bit controls the transfer of the listed architectural state to/from the respective fields in the UTCB (6.4) as follows:

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

MTD Bit	Access	Host Exception State	Guest Intercept State
POISON	w	Kills the EC	Kills the EC
GPR	rw	X0 ... X30	X0 ... X30
EL0_SP	rw	SP_EL0	SP_EL0
EL0_IDR	rw	TPIDR_EL0, TPIDRRO_EL0	TPIDR_EL0, TPIDRRO_EL0
A32_SPSR	rw	-	SPSR_ABT, SPSR_FIQ, SPSR_IRQ, SPSR_UND
A32_DACR_IFSR	rw	-	DACR, IFSR
EL1_SP	rw	-	SP_EL1
EL1_IDR	rw	-	TPIDR_EL1, CONTEXTIDR_EL1
EL1_ELR_SPSR	rw	-	ELR_EL1, SPSR_EL1
EL1_ESR_FAR	rw	-	ESR_EL1, FAR_EL1
EL1_AFSR	rw	-	AFSR0_EL1, AFSR1_EL1
EL1_TTBR	rw	-	TTBR0_EL1, TTBR1_EL1
EL1_TCR	rw	-	TCR_EL1
EL1_MAIR	rw	-	MAIR_EL1, AMAIR_EL1
EL1_VBAR	rw	-	VBAR_EL1
EL1_SCTLR	rw	-	SCTLR_EL1
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
EL2_HCR	rw	-	HCR_EL2
TMR	rw	-	CNTV_CVAL_EL0, CNTV_CTL_EL0 CNTKCTL_EL1, CNTVOFF_EL2
GIC	rw r	-	LR0 ... LR15 ELRSR, VMCR

6.7 Calling Convention

The following pages describes the calling convention for each hypercall. An execution context calls into the microhypervisor by loading the hypercall identifier and other parameters into the specified processor registers and then executes the `svc #0` instruction [1].

The hypercall identifier consists of the hypercall number and hypercall-specific flags, as illustrated in Figure 6.2.

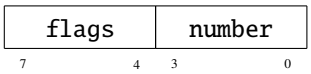


Figure 6.2: Hypercall Identifier

The status code returned from a hypercall has the format shown in Figure 6.3.

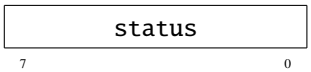


Figure 6.3: Status Code

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

IPC Call

$pt_{[63-8]}$ $hypercall_{[7-0]}$	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	$status_{[7-0]}$
own	X1		X1	≡
–	IP	svc #0	IP	IP+4

Create Execution Context

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

Create Scheduling Context

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

Create Portal

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

Create Semaphore

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

Control Protection Domain

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

Control Execution Context

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

Control Scheduling Context

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

Control Portal

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

Control Semaphore

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

Control Hardware

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

Assign Interrupt

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

Assign Device

$pd_{[63-8]}$ $hypercall_{[7-0]}$	X0	assign_dev ⟶	X0	$status_{[7-0]}$
ctx	X1		X1	≡
smg	X2		X2	≡
$sid_{[31-16]}$ $tbl_{[1-0]}$	X3		X3	≡
–	IP	svc #0	IP	IP+4

Part V

Appendix

A Acronyms

$\mathbf{ATTR_{CA}}$	Cacheability Attribute
$\mathbf{ATTR_{SH}}$	Shareability Attribute
CAP	Capability
$\mathbf{CAP_0}$	Null Capability
$\mathbf{CAP_{OBJ}}$	Object Capability
$\mathbf{CAP_{OBJ_{PD}}}$	PD Object Capability
$\mathbf{CAP_{OBJ_{EC}}}$	EC Object Capability
$\mathbf{CAP_{OBJ_{SC}}}$	SC Object Capability
$\mathbf{CAP_{OBJ_{PT}}}$	PT Object Capability
$\mathbf{CAP_{OBJ_{SM}}}$	SM Object Capability
$\mathbf{CAP_{MEM}}$	Memory Capability
$\mathbf{CAP_{PIO}}$	I/O Port Capability
CPU	CPU Number
EC	Execution Context
$\mathbf{EC_{CURRENT}}$	Current Execution Context
$\mathbf{EC_{ROOT}}$	Root Execution Context
ELF	Executable and Linkable Format
FPU	Floating Point Unit
HIP	Hypervisor Information Page
MTD	Message Transfer Descriptor
IP	Instruction Pointer
PD	Protection Domain
$\mathbf{PD_{CURRENT}}$	Current Protection Domain
$\mathbf{PD_{NOVA}}$	NOVA Protection Domain
$\mathbf{PD_{ROOT}}$	Root Protection Domain
PID	Portal Identifier
PT	Portal
QPD	Quantum Priority Descriptor
SC	Scheduling Context
$\mathbf{SC_{CURRENT}}$	Current Scheduling Context
$\mathbf{SC_{ROOT}}$	Root Scheduling Context
SEL	Capability Selector
$\mathbf{SEL_{EVT}}$	Event Selector Base
$\mathbf{SEL_{MEM}}$	Memory Capability Selector
$\mathbf{SEL_{OBJ}}$	Object Capability Selector
$\mathbf{SEL_{PIO}}$	I/O Port Capability Selector
SM	Semaphore

SP	Stack Pointer
SPC$_{\texttt{MEM}}$	Memory Space
SPC$_{\texttt{OBJ}}$	Object Space
SPC$_{\texttt{PIO}}$	I/O Port Space
TYPE$_{\texttt{SPC}}$	Space Type
TYPE$_{\texttt{TBL}}$	Table Type
UTCB	User Thread Control Block
VMM	Virtual-Machine Monitor
ipc_call	Hypercall: IPC Call
ipc_reply	Hypercall: IPC Reply
create_pd	Hypercall: Create Protection Domain
create_ec	Hypercall: Create Execution Context
create_sc	Hypercall: Create Scheduling Context
create_pt	Hypercall: Create Portal
create_sm	Hypercall: Create Semaphore
ctrl_pd	Hypercall: Control Protection Domain
ctrl_ec	Hypercall: Control Execution Context
ctrl_sc	Hypercall: Control Scheduling Context
ctrl_pt	Hypercall: Control Portal
ctrl_sm	Hypercall: Control Semaphore
ctrl_hw	Hypercall: Control Hardware
assign_int	Hypercall: Assign Interrupt
assign_dev	Hypercall: Assign Device

B Bibliography

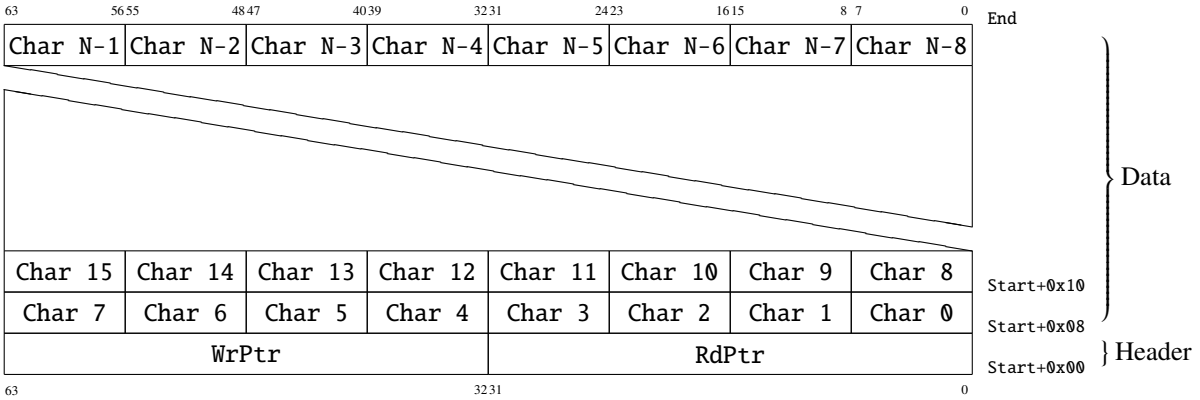
[1] *ARM Architecture Reference Manual ARMv8, for ARMv8-A Architecture Profile*. ARM Limited. URL https://developer.arm.com/docs/ddi0487/latest. Document Number: DDI0487. 37

[2] U. Steinberg and B. 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

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 is an in-memory data structure that consists of a header area and a data areas follows:



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

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

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

The fields of the header area are used as follows:

- `RdPtr` ranges from `0` ... `N-1`.
  It points to the **next** character that the console consumer will read and is typically advanced by the console consumer.
- `WrPtr` ranges from `0` ... `N-1`.
  It points to the **next** character that the NOVA microhypervisor will write and is only advanced by the NOVA microhypervisor.
- The console buffer is empty if `RdPtr` is equal to `WrPtr`.
- Otherwise `WrPtr` will be ahead of `RdPtr`, wrapping around the console buffer size `N` accordingly, i.e. character `N+x` will be stored in the same console buffer slot as character `x`.
- If the buffer becomes full, the NOVA microhypervisor will advance `RdPtr`, forcing the oldest character to be discarded from the console buffer.

C.2 UART Console

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

D Download

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

https://github.com/udosteinberg/NOVA