LynxSecure 6.3.0 API Guide

LynxSecure Release 6.3.0-rev16326

U.S. Patents 9,390,267; 8,745,745; 9,218,489; 9,129,123; 8,782,365; 9,208,030; 9,213,840.

Printed in the United States of America.

Contents

List of Figures

List of Tables

Preface

About this Guide

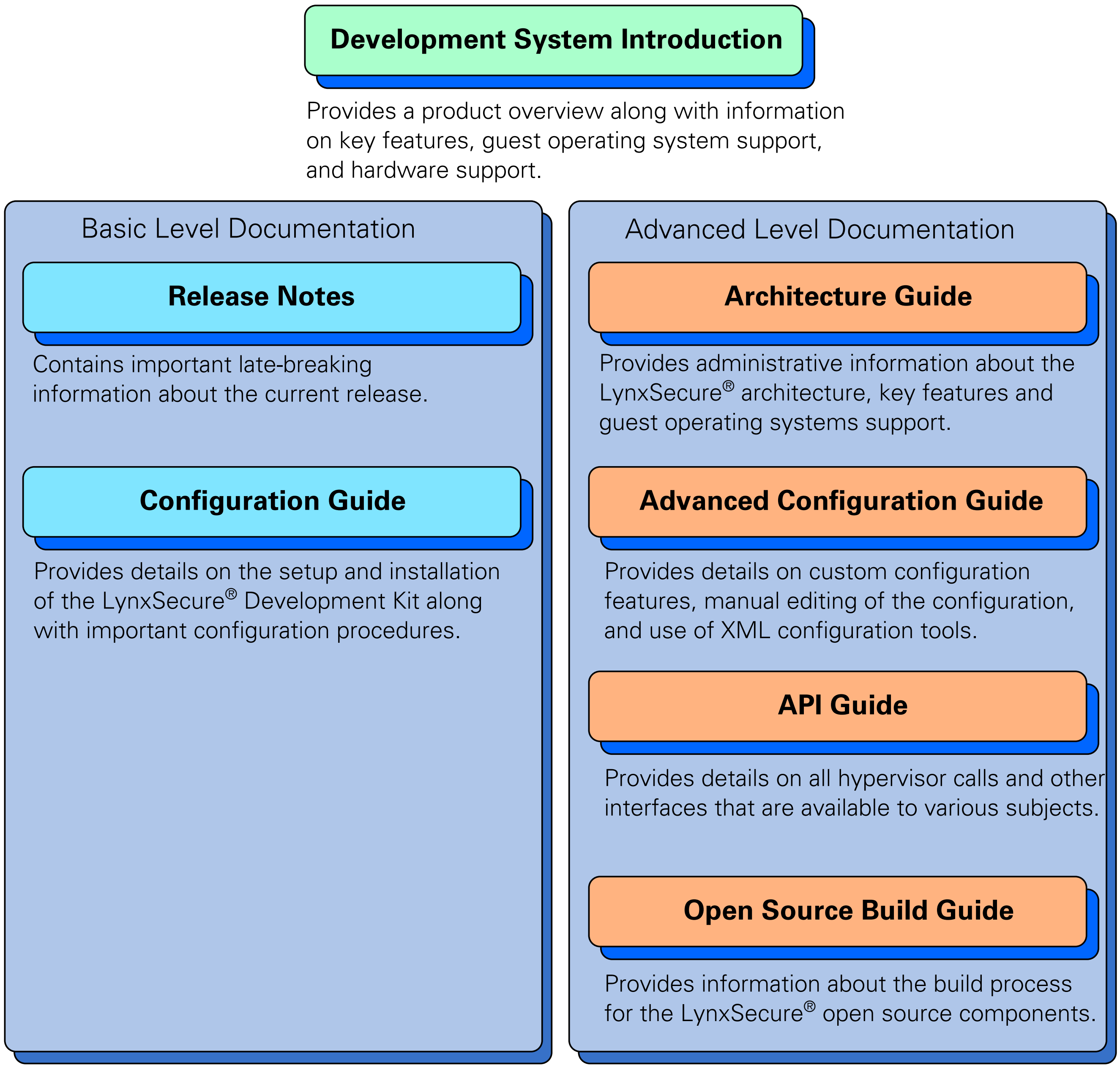
This guide, *LynxSecure® API Guide* lists all hypervisor calls available to subjects.

Intended Audience

The information in this guide is designed and written for system integrators and software developers who will build applications on top of LynxSecure and the available subjects. A basic understanding of Linux®/Unix® and familiarity with fairly complex configuration procedures are recommended.

For More Information

For more information on the features of LynxSecure, refer to the following printed and online documentation.



Typographical Conventions

The typefaces used in this manual, summarized below, emphasize important concepts. All references to file names and commands are case sensitive and should be typed accurately.

Font and Description	Examples
Times New Roman 10 pt. - Used for body text; *italicized* for emphasis, new terms, and book titles.	Refer to the *LynxSecure User's Guide*

Courier New 9 pt. - Used for environment variables, file names, functions, methods, options, parameter names, path names, commands, and computer data. Commands that need to be	`ls -l`
	`myprog.c`
	`/dev/null`
	`login:` **`myname`**

Font and Description	Examples
highlighted within body text, or commands that must be typed as-is by the user are **bolded**.	`#` **`cd /usr/home`**
Courier New Italic 9 pt. - Used for text that represents a variable, such as a file name or a value that must be entered by the user.	`cat` *`filename`* `mv` *`file1 file2`*

Courier New 7 pt. - Used for blocks of text that appear on the display screen after entering instructions or commands.	`Kernel: target1.srp`
	`> Loading: target1.srp`
	`> ......................`
	`> Booting`

Univers 45 Light Bold 8 pt. - keyboard options, button names, and menu sequences.	Enter, Ctrl-C

Special Notes

The following notations highlight any key points and cautionary notes that may appear in this manual.

NOTE: These callouts note important or useful points in the text.

CAUTION! Used for situations that present minor hazards that may interfere with or threaten equipment/performance.

Technical Support

Lynx Software Technologies, Inc. Technical Support is available Monday through Friday (excluding holidays) between 8:00 AM and 5:00 PM Pacific Time (U.S. Headquarters) or between 9:00 AM and 6:00 PM Central European Time (Europe).

The Lynx Software Technologies, Inc. World Wide Web home page [http://www.lynx.com] provides additional information about our products.

Lynx Software Technologies, Inc. U.S. Headquarters

Internet: `<support@lynx.com>`
Phone: (408) 979-3940
Fax: (408) 979-3920

Lynx Software Technologies, Inc. Europe

Internet: `<tech_europe@lynx.com>`
Phone: (+33) 1 30 85 06 00
Fax: (+33) 1 30 85 06 06

CHAPTER 1 *Overview*

This document, *LynxSecure® API Guide*, describes the interfaces provided by the hypervisor to the subjects and programming considerations for use of these interfaces.

The document is structured as follows:

- Chapter 1 provides an overview of the interfaces provided by the hypervisor.
- Chapter 2 describes the basic principles of subject operation.
- Chapter 3 describes the contents of the read-only page.
- Chapter 4 describes how the subject can make explicit requests for services ("hypercalls") and provides an overview of types of hypercalls.
- Chapter 5 lists all the supported hypercalls.
- Chapter 6 explains how subjects can access particular LynxSecure features.
- Appendix A lists hypercall permissions in each system state.
- Appendix B lists the audit event type identifiers for audit events generated by LynxSecure.

The api.h Header File

The `api.h` file defines the interfaces by which subjects explicitly interact with the LynxSecure SKH. The two primary ways of interaction are making hypercalls and obtaining information from the special memory areas that the SKH maps into the subject memory space.

NOTE: There are also implicit subject interactions with the SKH. For example, when a subject attempts to access an address for which it does not have an authorized flow in the configuration vector, then a software-like trap may implicitly cause execution to transfer from the subject to the SKH.

CHAPTER 2 Subject Operation

This chapter describes basic operation of the subject and how paravirtualized and fully virtualized subjects perform common tasks such as memory management, interrupt handling, etc.

Subject Initialization

When a subject is started, one of its memory regions is populated with the contents of a memory region of the type `BOOT`. The region to be initialized in this way is determined as described in section “Memory Flows and Memory Regions” in *LynxSecure 6.3.0 Advanced Configuration Guide*.

The copy is performed in the subject context. For that, LynxSecure® creates a memory region of the type `BOOTSTRAP` for each subject (unless it is already defined in the HCV), which contains the code that copies the boot image.

After the boot image is copied, execution continues at the address specified by the `startaddr` attribute in the configuration vector.

Subject Execution States

A subject can be in one of the following execution states at any given time:

- `SUBJECT_RUNNING`: All virtual processors of the subject are executing subject code.
- `SUBJECT_STOPPED`: All virtual processors of the subject are frozen. The only way the subject can leave this state is by starting the subject from the initial state. Any interrupts that arrive to a stopped subject are discarded.
- `SUBJECT_SUSPENDED`: All virtual processors of the subject are frozen. The subject may be resumed and its virtual processors would continue to execute subject code from the point at which they were previously frozen. Interrupts are accepted for delivery in this state and will be delivered when the subject eventually resumes.

The execution state of a subject can be changed by the subject itself (from Running to Stopped or Suspended) or by other subjects using hypercalls, provided that the operation is permitted in the configuration vector. LynxSecure hypercalls provide the following operations on the subject state: stop, start, suspend, resume and restart. The restart operation is equivalent to stopping, then starting the subject.

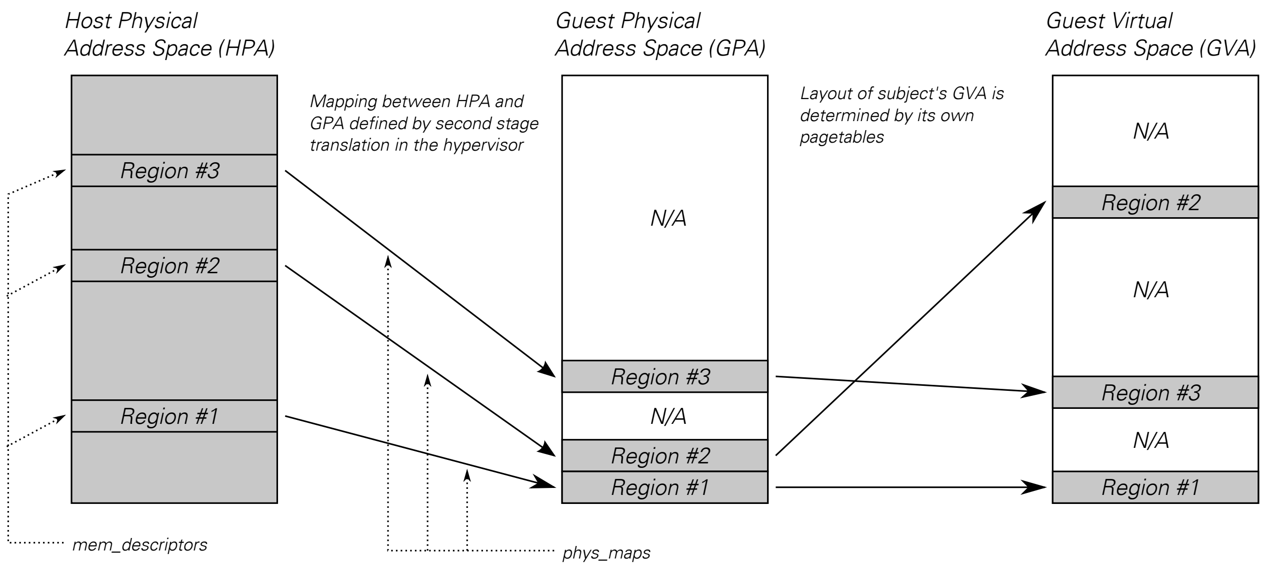
Subject Memory Address Space

A subject's physical memory address space (the Guest physical memory address space, or "GPA space" for short) is made up of the host memory regions assigned to the subject either directly or indirectly, and of memory ranges belonging to virtual devices. The host memory regions may be both host RAM and physical device I/O memory. The virtual device memory ranges have no representation in the host memory.

A fully virtualized subject's configuration in the HCV describes how its guest physical address space maps to the host physical address space. This mapping is reported to subject code in the `phys_maps` array in the RO page. A fully virtualized subject starts code execution with the virtual address translation (known as "paging" on the x86 platform and "the MMU" on the ARM platform) off.

A paravirtualized subject starts code execution with paging on and its configuration in the HCV describes how its initial virtual address space maps to its guest physical address space. That initial map is described in the `virt_maps` array in the subject's RO page. Afterwards, the subject may modify that page table or create its own page tables with custom mappings. Like in FV subjects, the `phys_maps` array describes the mapping of the guest to host physical address space, however it's not configurable in the HCV.

Figure 2-1: Address Space Construction in a Regular Subject



NOTE: On the x86 platform, it is possible for a fully virtualized subject to modify its GPA-to-HPA mapping from the original reflected in the RO page `phys_maps` array. For example, this is necessary to support relocatable PCI device I/O memory. LynxSecure virtualization components do this transparently to subject code when the subject programs PCI BAR registers. These changes do not update the `phys_maps` array.

Just like on bare hardware, the mapping from GVA to GPA is determined by subject's pagetables. If a subject turns paging off, its GVAs become identical to its GPAs.

The I/O memory of assigned physical devices is always "identity-mapped" in all types of subjects, meaning that its GPA ranges are identical to the corresponding HPA ranges. The user must be careful in order not to overlap those ranges with other memory region mappings.

The RAM layout in the GPA space can be either defined explicitly in the configuration, or generated automatically when the `ramsize` subject attribute is specified. The Autoconfig Tool uses that attribute for subjects by default; an explicit configuration can be generated with the `--explicit` option.

In regular subjects (those not "identity-mapped"), there is a default GPA layout that LynxSecure generates. For example, on all platforms, a few megabytes of GPA space just below the 4GB mark is allocated to the subject bootstrap code. On the x86 platform, the GPA space must also satisfy some legacy requirements, as guest software may break if the layout is not as expected of a legacy PC compatible system. Portions of this layout will be generated regardless of whether the `ramsize` subject attribute is used or not. If the attribute is used, LynxSecure will place as much RAM as possible in the low 4GB of the GPA space, and any RAM that doesn't fit below 4GB is placed at the GPA of 4GB (100000000 hexadecimal) and above.

The amount of address space available for RAM below 4GB is determined by the amount of address space occupied by device I/O memory and miscellaneous memory regions. On x86 systems, a large portion of the address space below

4GB is usually occupied by the PCI host bridge. The location and the size of that range depends on the configuration of the assigned physical/virtual devices, as well as the resources occupied by the corresponding devices and PCI host bridges in the HPA space. Therefore, it is hard to predict. LynxSecure attempts to minimize its size as much as possible without relocating any of the device I/O memory in the GPA space.

The miscellaneous memory regions placed below the bootstrap range include internal auxiliary data regions such as the subject's RO page. If no GPA is specified for those regions in the configuration, LynxSecure attempts to place them as high as possible below the subject bootstrap range.

Table 2-1: Regular Subject GPA Space Layout on the x86 platform

GPA Range (hex)	Size	Purpose
00000000..0009FFFF	640KB	Legacy low RAM (available to guest software)
000A0000..000BFFFF	128KB	Legacy physical or virtual VGA framebuffer
000C0000..000DFFFF	128KB	Legacy device Option ROM range
000E0000..000FFFFF	128KB	Legacy BIOS code and data
00100000 and up	Configurable	Available RAM
Below FFC00000	Varies	Device I/O and miscellaneous memory regions
FFC00000..FFFFFFFF	4MB	Subject bootstrap
100000000 and up	Configurable	Available RAM

Table 2-2: Regular Subject GPA Space Layout on the ARM platform

GPA Range (hex)	Size	Purpose
00000000 and up	Configurable	Available RAM
Below FFC00000	Varies	Device I/O and miscellaneous memory regions
FFC00000..FFFFFFFF	4MB	Subject bootstrap
100000000 and up	Configurable	Available RAM

Identity-Mapped Subjects

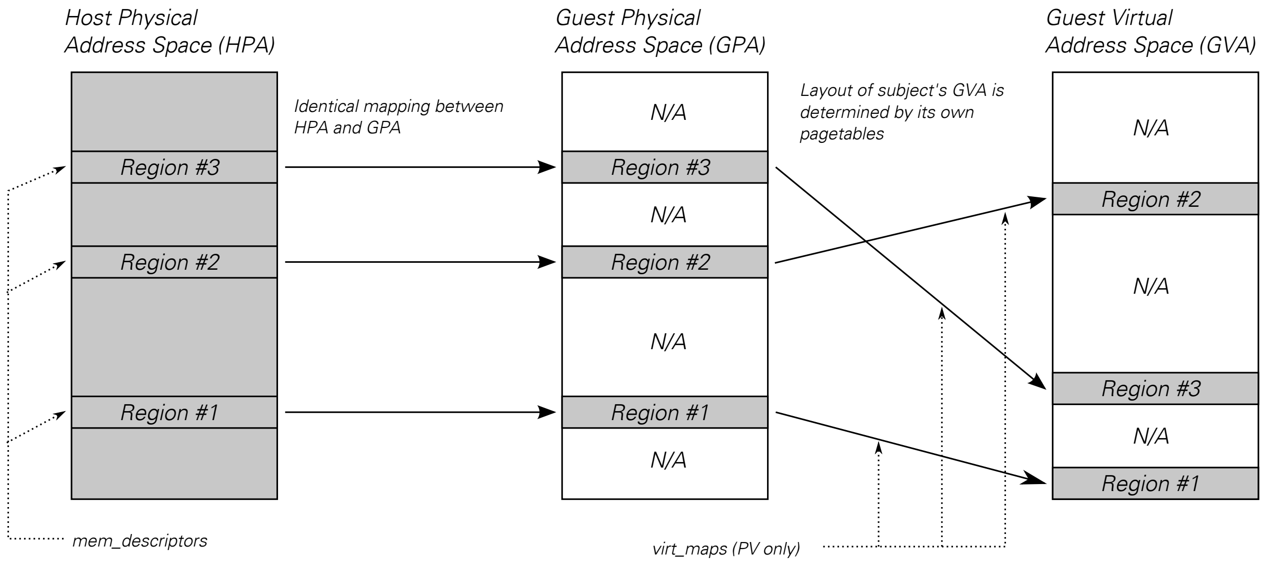
In an identity-mapped subject, the host memory regions are "identity-mapped" from HPA space to GPA space, meaning their GPA matches their HPA. Such a subject configuration enables physical device DMA on systems with no I/O MMU. Both PV and FV subjects can be identity-mapped, but only for PV subjects the identity-mapping is enabled automatically. Both PV and FV subjects may be configured for identity-mapping by explicitly setting RAM memory regions' GPA equal to their HPA in the HCV.

Identity mapping is automatically enabled for a subject by LynxSecure if all of the following is true:

- The system has no I/O MMU or it is disabled.
- The subject is a PV subject.
- The subject has assigned physical devices.

In an identity-mapped subject, RAM layout in GPA space may be unusual for the platform. Subject code must be able to accommodate such arbitrary RAM layouts.

Figure 2-2: Address Space Construction in an Identity-Mapped Subject



Modification of the Virtual Address Space

Both para- and fully virtualized subjects can modify their pagetable entries in the same way a natively run OS would, by writing to the page table entries. The hypervisor uses a hardware virtualization assist feature that adds a second stage to guest address translation that is performed every time subject code accesses memory. The second stage translates the GPA to the corresponding HPA, also applying any access mode restrictions specified in the configuration. If the access is permitted, it is carried out without hypervisor involvement. If the access is not permitted, it is trapped to the hypervisor and an audit record is created for the violation. Write accesses are then discarded and read accesses return undefined values to subject code.

Special Considerations for Accessing Memory Regions

For guest software running in a subject, in order to access a specific memory region assigned to that subject, the first step is to determine its GPA. Refer to section “Using the RO Page to Locate Memory Regions” (page 14). It is then necessary to map the memory region into the virtual address space. Each operating system has its own interfaces to perform such a mapping: in Linux®, it is `ioremap()` function; in LynxOS®, it is `permap()`, etc.

Interrupts

This section describes how interrupts are handled in subjects.

Interrupts in Paravirtualized Subjects

Whenever the hypervisor determines that there is an interrupt targeted to a paravirtualized subject and interrupt delivery is enabled in the destination virtual CPU of the subject, the interrupt is injected. The injected interrupt's vector is calculated as the sum of the incoming interrupt request (IRQ) and the subject's IRQ base.

The IRQ base is specified in the HCV using the `irqbase` subject attribute. The IRQ base cannot be changed at runtime.

Operations that involve the interrupt controllers (the Local APIC and/or the I/O APIC) in a natively running OS require hypercalls in a paravirtualized LynxSecure subject. When the subject completes processing of the interrupt, it must notify the hypervisor by issuing the HVCALL_INTR_EOI (page 29) hypercall. A subject can specify which of its virtual CPUs are to receive specific interrupts vectors by the means of the

HVCALL_ROUTE_INTR (page 33) hypercall. Within a subject, virtual CPUs can send interrupts to each other using the HVCALL_SEND_IPI (page 33) hypercall.

Similarly to the native APIC operation, a high-priority interrupt delivered to a paravirtualized subject's virtual CPU blocks lower priority interrupts on that virtual CPU until an End-Of-Interrupt (EOI) request is issued.

Interrupts in Fully Virtualized Subjects

In fully virtualized subjects, interrupts are processed by virtual interrupt controllers; no interrupt may bypass an interrupt controller and just arrive out of nowhere. The interrupt controllers are the same as the ones provided by the hardware platform: the 8259 PIC, the Local APIC and the I/O APIC.

Device-generated interrupts don't require any non-standard actions from subject software. Synthetic interrupts, such as those generated by a hypercall or by a message queue, do require additional actions. Unlike interrupts generated by a virtual or physical device, synthetic interrupt sources are not reported to guest software via ACPI tables or via some device's PCI configuration space. Therefore, the GuestOS doesn't know anything about synthetic interrupt sources automatically. The GuestOS needs a custom driver that expects the synthetic interrupt and registers a handler for it. To do that, the driver needs to determine the virtual interrupt controller input (IRQ line for the 8259, GSI pin for the I/O APIC) activated by the interrupt. That information can be obtained using the interface described in Table 2-3 (page 7).

Configuration note: if the configuration specifies an IRQ associated with a synthetic interrupt, it is not the IRQ that the subject is going to see when the interrupt arrives. The configuration IRQ is an internal identifier (the synthetic interrupt vector) for the interrupt; it goes through an additional layer of virtualization and is converted to the virtual interrupt controller input number that is activated when the interrupt arrives.

Synthetic interrupts behave as edge-triggered interrupts: they don't need any acknowledgment other than a regular End-of-Interrupt to the virtual interrupt controller.

Table 2-3: Interrupt Information I/O Ports

Port	Access Mode	Description
601h	8-bit R/W	The vector selector: writing an internal vector number to this port selects the vector; reading from the ports below returns information about that vector. The internal vector number is calculated as follows: add the configured IRQ of the synthetic interrupt (or, if automatically determined, its IRQ according to the RO page) to the value of the `irq_base` field in the subject's RO page.
605h	8-bit R/O	The virtual 8259 input index (the legacy IRQ) for the selected vector.
606h	16-bit R/O	The virtual I/O APIC input index (the Global System Interrupt, GSI) for the selected vector.

NOTE: This interface is non-standard and may change in future releases.

Configuring FVS boot parameters at run time

By default LynxSecure includes the necessary configuration to boot a given subject as part of the SRP. If run time control over this process is needed, there is a mechanism that allows for the subject boot image and configuration to be stored and loaded separately from the SRP.

To detach the subject boot image from the SRP, the subject in question must specify a bootregion as described in the config guide. With the bootregion option specified, instead of including the boot image in the SRP, it is instead assumed that the boot image and a boot trailer will be preloaded into ram at the specified location before LynxSecure is started. The boot trailer describes the boot image that was preloaded before LynxSecure was executed.

The boot trailer is described by the following C struct:

```
/** Firmware trailer */
typedef struct firmware_trailer_s {
 uint32_t magic;     ///< Should always be set to FIRMWARE_TRAILER_MAGIC
 uint32_t flags;     ///< Contains the version of the header, and flags indicating the presence of fields.
 uint32_t csum;      ///< flags + csum == magic
 uint32_t size;     ///< Firmware image size
 uint64_t addr;     ///< Firmware image source address
 uint64_t dst;     ///< Firmware image destination address
 uint64_t jump_addr; ///< Firmware jump address
} __attribute__((packed,aligned(8))) firmware_trailer_t;
/** Firmware trailer magic, ASCII "LSKB" as a little endian uint32 */
#define FIRMWARE_TRAILER_MAGIC (0x4c534b42)

/** bit 0-15 are used as flags, bit 16-31 are used as version info */
#define FIRMWARE_TRAILER_VERSION (0x0001 << 16)
#define FIRMWARE_TRAILER_VERSION_MASK 0xFFFF0000

/**
 (struct firmware_trailer_s).flags bits 0-15 are used to indicate features
*/
enum {
 FW_HAS_SIZE     = 1 << 0, ///< Set this bit if the size field is populated
 FW_HAS_ADDR     = 1 << 1, ///< Set if the addr field is populated
 FW_HAS_DST      = 1 << 2, ///< Set if the dst field is populated
 FW_HAS_JUMPADDR = 1 << 3, ///< Set if the jump addr is populated
 FW_IN_PLACE     = 1 << 4, ///< Set if firmware should be executed in place
};
```

The fields of this struct should be populated as follows:

Table 2-4: Boot Trailer Contents

Field	Value
magic	This should always be set to FIRMWARE_TRAILER_MAGIC
flags	The upper 16 bits of this field are used to indicate the version of the trailer. The Lower 16 bits hold flags. If any of the fields in this table below are set, that is indicated by a flag in this field. Additionally if it is desired that the boot image be executed in place, that is also indicated here. Note: If the guest is to be configured with the FW_IN_PLACE flag, then extra care should be taken with the configuration of the target memory region. Specifically, the memory region should be specified as PROGRAM, and the subject should have a memory flow for this region that enables the write permission. In autoconfig bootregions, this corresponds to the RW_RAM mode.
csum	The csum should always equal the magic minus the flags.
size	This field describes the size of the boot image. The corresponding bit must be set in the flags field.
addr	This field describes the start address of the boot image. The corresponding bit must be set in the flags field.
dst	This field describes the destination address of the boot image. The corresponding bit must be set in the flags field.
jump_addr	This field describes the jump address of the boot image. The corresponding bit must be set in the flags field.

Included with LynxSecure is a tool called `lsk_boot_trailer` that can be used to generate the boot trailer datastructure above. Consult `lsk_boot_trailer --help` for usage information.

During boot before LynxSecure is executed, the boot image should be placed at the address specifed in the HCV. The boot trailer should be placed at the beginning of the final page in the specified memory region. For example, if a memory region is specifed to start at `0x10000000` and to be of size `16M`, then the boot trailer should be located at `0x11000000`. Note that Autoconfig Tool will implicity allocate an extra page to store the boot trailer.

Once the boot image and trailer are in place, LynxSecure can be executed. During boot, LSK will detect the boot trailer and use the information contained to override the configuration built into the SRP.

CHAPTER 3 *Read-Only and Argument Pages*

For each subject, there is a special read-only page ("RO page" hereinafter) used to perform introspection. Subject may also have a read-only page to pass arguments to the subject, so called ARG page. The Chapter 2, "*Configuration Vector Organization*" in *LynxSecure 6.3.0 Advanced Configuration Guide* describes how RO and ARG pages are configured.

Read-Only Page

Each subject has one and only one unique RO page. The RO page may not be written to by the subject (or any subject), only by the SKH.

The data structure for each subject conveys information necessary for paravirtualization such as accessible memory descriptors, accessible memory maps, named object maps, available devices, ticks per second and TSC ticks between the timer ticks and interrupt information. It also contains information about all devices the subject has access to, the device memory and I/O regions (BARs), the device IRQs that the subject may use instead of reading from hardware. Some of this information may also be useful for a fully virtualized subject.

Locating RO Page

There are two ways of locating the RO page. The preferred way is that the subject can issue the HVCALL_GET_ROPAGE_INFO (page 27) hypercall. The example below locates the RO page using the hypercall:

```
const ls_ro_page_t *
get_ropage(void)
{
    ropage_info_t info;

    VMCALL(HVCALL_GET_ROPAGE_INFO, (uintptr_t)&info);
    return (const ls_ro_page_t *)(uintptr_t)info.initial_addr;
}
```

The `initial_addr` field in the `ropage_info_t` structure reports the address at which the RO page is initially accessible to the subject. For paravirtualized subjects, guest virtual address (GVA) is reported. For fully virtualized subjects, guest physical address (GPA) is reported. Therefore, the code above would work in a paravirtualized subject, or in a fully virtualized subject with the "identity map" (i.e. when virtual addresses are the same as physical). Otherwise, fully virtualized subjects need to take additional steps (depending on the OS type) to map the RO page into virtual address space. For example, FV Linux® subject would need to call `ioremap(info.paddr)` to obtain the virtual address.

NOTE: Paravirtualized subjects can remap the RO page to a different virtual address. The `initial_addr` would still report the original virtual address, which can no longer be relevant.

The `ropage_info_t` structure also reports the physical address at which RO page resides as the `paddr` member, and RO page size as the `size` member.

Alternatively, if the address space layout for the subject is known at the compile time, the address of the RO page can be hardcoded in the subject. This approach is discouraged, as it is prone to misconfiguration if the address space configuration changes.

Parsing RO Page

The layout of the RO page is defined in the `ls_ro_page_t` structure in the `api.h` header.

The RO page also contains some information that may not be interpreted by a subject directly, e.g. data for the platform emulation framework. Any fields not described explicitly in the sections below should be treated as reserved.

Scalar Fields

Table 3-1 (page 12) describes the scalar fields in the RO page.

Table 3-1: Scalar Objects in the RO Page

Member	Description
`api_id`	LynxSecure® Application Interface identifier. Refer to section “LynxSecure API Version” (page 13) for details.
`release_version`	String representation of LynxSecure release version.
`name`	This subject's name.
`subject_id`	Subject's unique identifier, as defined by the configuration. Note that this value is different from the exported resource identifier for the subject that audit records use.
`subjects_num`	Total number of subjects defined in the configuration.
`subject_type`	One of the `ls_subject_type_t` values which describes the subject's type: a fully virtualized, or a 32-bit or 64-bit paravirtualized subject.
`subject_role`	One of the `ls_subject_role_t` values which describes the role of the subject: a Virtual Device Server, an Initiated or Continuous Built-In Test, a LynxSecure Application (a small memory footprint subject), or Other (a generic subject, possibly running a Guest Operating System). A subject's role is a hint for automatic configuration software.
`cpus_num`	Number of virtual CPUs assigned to the subject.
`ticks_per_sec`	Number of scheduler interrupts ("ticks") per second.
`tb_per_tick`	Number of TSC (timestamp counter) increments per scheduler tick.
`tb_per_sec`	Number of TSC increments per second.
`irq_base`	The subject's IRQ base (the interrupt vector corresponding to IRQ #0).
`irq_sct`	The IRQ of the interrupt delivered at each System Clock Tick, or `NO_IRQ` if none is delivered.
`irq_minor_frame`	The IRQ of the interrupt delivered at each scheduling minor frame, or `NO_IRQ` if none is delivered.
`irq_audit`	The IRQ of the interrupt delivered at each Audit event, or `NO_IRQ` if none is delivered.
`used_pool`	For paravirtualized subject, this array describes the number of pages used in each page table region for the initial mapping. The array is indexed by the `pool_type_t` enumeration values, such as `PT_PML4`. Note that future releases of LynxSecure may require preservation of the initial mapping and will reject any attempt to modify the pages used for the initial mapping.
`ro_page_size`	Total size of the RO page, in bytes.
`percpu_timer_per_sec`	Number of Local APIC counter increments per second.
`san_mode`	Sanitization execution context, one of `LS_SAN_INACTIVE` (normal subject execution mode) or `LS_SAN_ACTIVE` (sanitization mode).
`subject_flags`	Subject state flags: **`SUBJECT_FLAG_RESTARTED`** Subject has been restarted after the original start-up.

Member	Description
	SUBJECT_FLAG_PCI_MCFG Subject has PCIE platform feature assigned and the platform supports PCI Express. The information in pci_mcfg_info field is valid. **SUBJECT_FLAG_NO_UNREAL_MODE** Subject has no support for x86 Unreal Mode.
pci_mcfg_info	Structure containing the location of the PCI Express memory-mapped configuration area (MCFG).

Arrays

The RO page defines several object arrays. Such arrays are described in `ls_vlist` structure in `api.h`. The `ls_vlist` structure contains the offset of the array from the start of the RO page and the number of elements in the array. To access the objects in such arrays, code must know the type of the objects in the array. In the example below, `object_t` is the type of the objects contained in the array and `vlist` is the member of the `ls_ro_page_t` structure describing array of these objects.

```
const object_t *
get_obj_by_index(const ls_ro_page_t *ro, unsigned int idx)
{
    return idx >= ro->vlist.count ? NULL :
        (const object_t *)((uintptr_t)ro + ro->vlist.offset) + idx;
}
```

Table 3-2: Object Arrays in the RO Page

Member	Object Type	Description
mem_descriptors	rmr_t	Memory descriptors representing a contiguous range of physical memory accessible to this subject.
phys_maps	ls_mem_map_t	The initial layout of the subject's GPA space, composed of GPA mappings of physical memory and virtual device I/O memory. Note that this is only the initial mapping and therefore may not be accurate if the subject s FV and has modified its GPA space (for example, by reprogramming some PCI device BARs). The GPA space is static in PV subjects.
virt_maps	ls_mem_map_t	The initial layout of the subject's GVA space, composed of GVA mappings of the physical mappings contained in the phys_maps array. This array is only used in PV subjects. Note that this is only the initial mapping and therefore may not be accurate if the subject has modified its page tables.
msg_channels	ls_msgchn_t	Message channel description, such as subject IDs for sending and receiving subjects, message size, maximum number of messages in queue, synthetic interrupt for this subject, etc.
devices	struct s_device	Devices assigned to the subject.
vdevices	struct s_vdevice	Virtual (emulated) devices assigned to the subject.
host_bridges	ls_host_bridge_t	Physical host-to-PCI bridge information.

LynxSecure API Version

Before LynxSecure release 5.1, the API ID consisted of two 64-bit integers at the beginning of RO page. The first one identified LynxSecure release, the second one identified the API revision within that release.

Starting with LynxSecure release 5.1, the API ID has been changed to the structure defined as `ls_api_id_t` in the `guestos/api.h` header. To validate the API version that your application was compiled against, which is the most typical usage of the API ID, it is sufficient to compare the structure in the RO page with a local, compiled-in, version initialized using `LYNXSECURE_API_ID` macro.

For less common purposes (such as conditionally compiling the code for different releases of LynxSecure), it may be necessary to understand how the numbers in the `ls_api_id_t` are produced. The first field of the structure, `main`, determines the version of the API as it progresses in the course of mainstream LynxSecure development (so called "trunk"). Changes to the subject API, be it an addition of a new hypercall or a change in the RO page layout, are reflected in this field. Another field, `nsubs`, determines the number of the elements in the `sub` array. Each element in this array uniquely identifies the branch (in the `branch` field) and the API revision on the branch (in the `rev` field).

With this numbering scheme, it is possible to see if any given version is older, newer, or unrelated to another version of API.

Using the RO Page to Locate Memory Regions

The RO page contains the description of the address space of the subject in the `mem_descriptors`, `phys_maps` and `virt_maps` arrays (the latter is only used in PV subjects). See section "Subject Memory Address Space" (page 3) for explanation of the differences between memory space in paravirtualized and fully virtualized subjects.

If one needs to find the GPA of a memory region, the first step is to find the memory descriptor for the memory region of interest. To perform lookup based on the region's name, one iterates over the `mem_descriptors` array containing `rmr_t` structures, which describe physical memory regions assigned to the subject. For each entry, the `name` field can be compared with the sought memory region's name.

To find a region of a desired type, iterate over the `mem_descriptors` array and test the `memory_type` field for the desired memory type bits set.

To find the GPA of a memory region, the `phys_maps` array containing `ls_mem_map_t` structures must be searched. If the `res_qual` field is `PHYSMAP_MEMREG`, the memory mapping is one of a physical memory region and the `res_idx` field contains the index of the mapped memory region in the `mem_descriptors` array. Search for the entry that references the previously found descriptor. The `addr` field in that entry contains the GPA of the memory region and the `size`field contains the size of the GPA mapping, which could be equal to or less than the size of the mapped memory region.

A reverse translation from a GPA to the corresponding memory region can be done by doing the look-up procedure described above in the reverse order: first find the GPA mapping entry by matching the desired GPA against the mapping's address and size, then find the corresponding memory region from the `res_qual` and `res_idx` fields in that entry. Note that if `res_qual` contains a value other than `PHYSMAP_MEMREG`, the mapping maps virtual device memory rather than a physical memory region. In that case, the `res_qual` fields contains the virtual device index in the `vdevices` array, and the `res_idx` field contains that virtual device's resource index in the `resources` array of the `ls_vdevice_t` structure that is mapped.

PV subjects start with the virtual address translation active, meaning that the GVA space is the active address space when the subject starts executing its code. The initial layout of the PV subject's GVA space is described in the `virt_maps` RO page array. The array contains `ls_mem_map_t` structures. Each entry describes a mapping of a GPA mapping into the GVA space. The `res_idx` field contains the index of the physical mapping in the `phys_maps` array, the `addr` field contains its GVA, and the `size` field contains the size of the GVA mapping, which could be equal to or less than the size of the GPA mapping.

NOTE: The RO page memory map arrays describe the *initial* state of the address space after the subject is started or restarted. A fully virtualized subject may make minor changes to its GPA space; for example, by re-programming its PCI device BARs. Similarly, a PV subject may modify its page tables and change how its GVA space is mapped to its GPA space. Those changes are *not* reflected in the RO page memory map arrays.

Argument Page

The purpose of the ARG page is to provide a special page of memory by which boot arguments may be passed to a subject. For example, one can use it to specify Linux kernel boot parameters for a PV Linux subject. The contents of subjects' ARG pages are set in the configuration vector.

The argument page (ARG page) is similar to the RO page, in that only one ARG page could be specified for a given subject in the XML configuration file.

To locate the ARG page, one needs to iterate over the `mem_descriptors` array in the RO page to find the `rmr_t` structure with the type set to `MEM_TYPE_ARGPAGE`. Once such region is found, one needs to use the procedure described in section “Using the RO Page to Locate Memory Regions” (page 14) to find out the physical and/or virtual address where ARG page can be accessed.

CHAPTER 4 *Overview of Hypervisor Calls*

A hypervisor call (hypercall for short) is a request for service sent by code running in a LynxSecure® subject to the LynxSecure hypervisor. LynxSecure hypervisor provides a range of services such as subject execution state control, audit record management, time keeping, etc.

Restrictions on Hypercalls

Some hypercalls are restricted by the policy flows defined in the HCV. Refer to the section "<subject> — Subject" in *LynxSecure 6.3.0 Advanced Configuration Guide* for additional information on configuration of hypercall access.

Availability of certain hypercalls depends on the current system state. The section "System State Manager (SSM)" in *LynxSecure 6.3.0 Architecture Guide* describes the system states in detail; the Appendix A, "*Hypercall Permission by System State*" (page 57) lists which hypercalls are permitted in each system state.

To preserve subject OS security model, LynxSecure does not allow subject code executing with user privileges (ring 3) to issue hypercalls. Attempt to issue hypercall from ring 3 will return with the `LYNXSK_PDE_PERMISSION_DENIED` error code.

Using the Hypercalls with GCC

In LynxSecure, hypercalls take the hypercall number (identifying the requested service) and up to three 64-bit arguments, depending on the hypercall number.

LynxSecure development environment provides a header file, `api.h`, that contains hypercall numbers defined as symbolic macros.

The `api.h` header also provides a convenience macro, `VMCALL`. This macro uses GCC assembler constraints to describe the argument passing conventions. Using this macro, it is possible to issue hypercalls as if they were function calls.

NOTE: The `VMCALL` macro assumes all its arguments to be 64-bit integers. If a hypercall takes a pointer as an argument, that pointer needs to be cast to a 64-bit integer.

Examples in Chapter 5, "*Hypercalls Alphabetic Listing*" (page 23) assume that the GCC compiler is used.

Using the Hypercalls with Other Compilers

This section describes the low-level mechanisms of hypercall operation.

In Intel® architecture, a hypercall is performed by issuing a dedicated assembly instruction, `VMCALL`. In case the compiler being used does not support this instruction, its bytecode is `0F 01 C1`. This instruction does not take any arguments; thus, arguments to hypercall are passed via registers. There are two argument passing conventions employed by LynxSecure; one is selected based on subject's current operating mode.

Table 4-1: Hypercall Argument Passing Convention

	64-bit subject	32-bit subject	
		MS 32 bits	LS 32 bits
Hypercall number	RCX	n/a	ECX
Argument 1	RDI	EDX	EAX
Argument 2	RSI	EDI	ESI
Argument 3	RDX	EBP	EBX
Return value	RAX	EDX	EAX

Hypercalls Taking a Memory Address

There are a number of hypercalls in LynxSecure which take Guest Virtual Address (GVA) arguments. GVA is an address the subject uses to address memory in its current MMU mode (for example, in the proprietary address space of a GuestOS process), as opposed to the Guest Physical Addresses (GPA) which are the locations in guest physical memory address space. Early versions of LynxSecure had a set of hypercalls taking GPAs which have been removed starting with LynxSecure version 4.0. With a GPA hypercall, it was necessary for the subject code to translate virtual addresses to physical addresses before passing them to the hypervisor. With the GVA hypercalls, one can simply pass the virtual address obtained, for example, with the C language & (ampersand) operator, and LynxSecure will perform the necessary translation. The GVA hypercalls will also accept and correctly interpret GPA addresses passed from code executing in physical address mode (with paging disabled) in fully virtualized subjects.

Whenever an address is passed to the LynxSecure hypervisor as a hypercall argument, it is always to have the hypervisor either read some data from or write some data to memory at that address. LynxSecure checks the permissions specified in subject-local MMU structures, such as page tables, for every address passed as a hypercall argument. Note that only certain types of memory are considered valid destination addressed. The valid memory types are: PROGRAM, SHM, ROPAGE, VIRTUAL_IO, SANSRC, SANDST, and BOOTSTRAP. All other memory types will be rejected. If the operation requested by the subject is not allowed by its local MMU structures, the hypercall fails. If the operation succeeds, LynxSecure updates the local MMU structures; for example, it sets the "dirty" and "accessed" bits in the subject local page tables, as necessary.

Note that LynxSecure does not apply x86 segmentation to the virtual addresses it receives from the subject as hypercall arguments; all addresses are assumed "flat". Typically, this is not a concern for C/C++ code. However, some GuestOS kernels use FS and GS register based segmentation for certain data. It is the subject code responsibility to translate segmented addresses to the flat format before passing them to LynxSecure.

NOTE: A common pitfall when using the `VMCALL` macro in 32-bit subjects is to cast 32-bit pointer to a signed integer type, such as `int` or `long`. According to C language integer type promotion rules, a value cast in this way would get sign-extended to 64 bits, not zero-extended. The resulting address would be invalid for 32-bit subject and will be rejected by the hypervisor with the error code `LYNXSK_ADDRESS_FAULT`.

Using Hypercalls from FV Subjects

By default, fully virtualized subjects are prohibited from using hypercalls. Attempt to issue a hypercall from a subject which has not been granted such permission will result in invalid opcode exception being delivered to the subject.

Hypercall invocation privilege may be granted to trusted subjects by setting the `guesthypercallperm` attribute to `ALLOW` in subject's description in the HCV.

Some of the hypercalls are not allowed in a fully virtualized subjects even if hypercalls are enabled in that subject. Refer to section “Hypercalls Allowed in Fully Virtualized Subjects” (page 19) to determine if a particular hypercall is allowed in fully virtualized subjects.

Hypercalls Allowed in Fully Virtualized Subjects

- `HVCALL_SYNTH_INTR`
- `HVCALL_TIME_GET_MONOTONIC`
- `HVCALL_TIME_GET_V`
- `HVCALL_TIME_SET`
- `HVCALL_TCAL_GET_V`
- `HVCALL_TCAL_SET`
- `HVCALL_TIME_LEFT_MNR_V`
- `HVCALL_MSG_RECV_V`
- `HVCALL_MSG_SEND_V`
- `HVCALL_SESM_STOP_SUBJECT`
- `HVCALL_SESM_RESTART_SUBJECT`
- `HVCALL_SESM_SUSPEND_SUBJECT`
- `HVCALL_SESM_START_SUBJECT`
- `HVCALL_SESM_RESUME_SUBJECT`
- `HVCALL_SESM_GET_STATE_V`
- `HVCALL_RETRIEVE_AUDIT_RECORD_V`
- `HVCALL_RETRIEVE_OVERFLOW_AUDIT_RECORD_V`
- `HVCALL_STORE_AUDIT_RECORD_V`
- `HVCALL_CHANGE_SCHD_POLICY`
- `HVCALL_GET_SCHD_POLICY_V`
- `HVCALL_FLEX_SCHD_GIVE_ALL`
- `HVCALL_FLEX_SCHD_GIVE_UNTIL_SCT`
- `HVCALL_FLEX_SCHD_GIVE_UNTIL_EVENT`
- `HVCALL_FLEX_SCHD_RETURN`
- `HVCALL_SSM_GET_STATE_V`
- `HVCALL_SSM_SET_LAST_STATE`
- `HVCALL_SSM_SET_MAINTENANCE_INSECURE`
- `HVCALL_SSM_SET_MAINTENANCE_SECURE`
- `HVCALL_SSM_SET_INITIATED_BIT`
- `HVCALL_SSM_SET_OPERATION`
- `HVCALL_SSM_SET_RESTART`
- `HVCALL_SSM_SET_SHUTDOWN`

- `HVCALL_GET_ROPAGE_INFO`

A Functional Synopsis of the Hypercalls

The hypercalls can be grouped into functional classes. What follows is a quick synopsis of each of the hypercalls, grouped by function, followed by a detailed reference of the hypercalls, organized alphabetically.

Interrupts

HVCALL_INTR_EOI (page 29)

Report the End Of Interrupt processing to the hypervisor.

HVCALL_ROUTE_INTR (page 33)

Specifies how an external interrupt is routed to VCPUs in the subject.

HVCALL_SEND_IPI (page 33)

Sends an inter-processor interrupt to another VCPU in the same subject. This call can also halt and start other VCPUs in the same subject.

HVCALL_SYNTH_INTR (page 49)

Inject a synthetic interrupt with the specified IRQ number into the target subject.

HVCALL_VDEV_NOTIFY_PEER (page 54)

Send a notification to the peer subject of a virtual device.

Internal Time Keeping

HVCALL_TIME_GET_MONOTONIC (page 51)

Reads the Monotonic time.

HVCALL_TIME_GET_V (page 52)

Reads the Wall time.

HVCALL_TIME_SET (page 53)

Sets the Wall time (the Wall time is the current time in seconds since the epoch midnight January 1st, 1970).

HVCALL_TCAL_GET_V (page 50)

Returns the Wall time timebase calibration value.

HVCALL_TCAL_SET (page 50)

Sets the Wall time timebase calibration value.

HVCALL_TIME_LEFT_MNR_V (page 53)

Returns the number of Monotonic time nanoseconds left in the current minor scheduling frame.

Inter Subject Messaging

HVCALL_MSG_RECV_V (page 29)

Reads the next 64-byte message from a message channel.

HVCALL_MSG_SEND_V (page 30)

Writes a 64-byte message to a message channel.

Subject Execution State Manager (SESM)

HVCALL_SESM_STOP_SUBJECT (page 37)

Stops a running subject.

HVCALL_SESM_RESTART_SUBJECT (page 35)

Restarts the identified subject from its original start address.

HVCALL_SESM_SUSPEND_SUBJECT (page 38)

Suspends the target subject.

HVCALL_SESM_START_SUBJECT (page 37)

Starts a stopped subject from its original start address.

HVCALL_SESM_RESUME_SUBJECT (page 36)

Resumes a suspended subject.

HVCALL_SESM_GET_STATE_V (page 34)

Returns the state of the target subject.

Audit

HVCALL_RETRIEVE_AUDIT_RECORD_V (page 31)

Retrieves an audit record from the audit buffer.

HVCALL_RETRIEVE_OVERFLOW_AUDIT_RECORD_V (page 32)

Retrieves an overflow audit record from the audit buffer.

HVCALL_STORE_AUDIT_RECORD_V (page 47)

Stores audit events in the LynxSecure audit log.

Scheduling Policy

HVCALL_CHANGE_SCHD_POLICY (page 23)

Selects a new scheduling policy.

HVCALL_GET_SCHD_POLICY_V (page 28)

Returns the identifier of the currently active scheduling policy.

Flexible Scheduling

HVCALL_FLEX_SCHD_GIVE_ALL (page 24)

Donates time slice to another subject indefinitely.

HVCALL_FLEX_SCHD_GIVE_UNTIL_SCT (page 26)

Donates the rest of the current clock tick to another subject.

HVCALL_FLEX_SCHD_GIVE_UNTIL_EVENT (page 24)

Donates the current time slice to another subject until an asynchronous event.

HVCALL_FLEX_SCHD_RETURN (page 27)

Return the time slice to the scheduled owner.

System Management

HVCALL_SSM_GET_STATE_V (page 42)

Returns the current SSM state.

HVCALL_SSM_SET_LAST_STATE (page 43)

Transitions the system to the last state entered before the current state.

HVCALL_SSM_SET_MAINTENANCE_INSECURE (page 44)

Initiates a transition to the Maintenance Insecure mode.

HVCALL_SSM_SET_MAINTENANCE_SECURE (page 45)

Initiates a transition to Maintenance Secure mode.

HVCALL_SSM_SET_INITIATED_BIT (page 43)

Initiates Built-In Test (BIT).

HVCALL_SSM_SET_OPERATION (page 45)

Initiates a transition to Operation mode.

HVCALL_SSM_SET_RESTART (page 46)

Initiates a restart of the system.

HVCALL_SSM_SET_SHUTDOWN (page 46)

Initiates a shutdown of the system.

High-Resolution Timers

HVCALL_SET_HRTIMER (page 39)

Schedules the High-Resolution Timer expiration.

HVCALL_SET_HRTIMER_VEC (page 40)

Sets the High-Resolution Timer interrupt vector.

Debugging

HVCALL_SUBJECT_LOG (page 48)

Prints a message on the hypervisor's consoles.

HVCALL_SKDB_ENTER (page 40)

Make the hypervisor drop into the built-in debugger.

HVCALL_SKDB_EXECUTE (page 41)

Execute SKDB commands from memory buffer and save their output into another memory buffer.

Miscellaneous

HVCALL_GET_ROPAGE_INFO (page 27)

Obtains RO page information such as its address and size.

HVCALL_STROBE_WATCHDOG (page 48)

Strobes subject watchdog.

CHAPTER 5 *Hypercalls Alphabetic Listing*

This chapter lists all hypervisor calls available to subjects.

CHANGE_SCHD_POLICY - Select New Scheduling Policy

Synopsis

```
retval = VMCALL(HVCALL_CHANGE_SCHD_POLICY, policy_id);
```

Description

Selects a new scheduling policy.

Arguments

Argument Name	Description
`uint32_t policy_id`	The new policy identifier.

Permissions

```
<subject … writeschedpolicyperm="ALLOW" …/>
```

Notes

- The scheduling policy must be defined as a `schedulingpolicy` element in the XML configuration vector.
- Passing the identifier of the `sibitschedpolicy` or `maintschedpolicy` policy results in an error being returned as only the SKH can directly select these policies. A subject can indirectly select the `sibitschedpolicy` policy via the `HVCALL_SSM_SET_INITIATED_BIT` hypercall (page 43), and the `maintschedpolicy` policy via the `HVCALL_SSM_SET_MAINTENANCE_SECURE` hypercall (page 45).
- The new scheduling policy becomes effective when the current major frame completes.

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded.
`LYNXSK_PDE_PERMISSION_DENIED`	The calling subject has no permission to perform the operation.
`LYNXSK_MSP_CHANGE_ALREADY_IN_ PROGRESS`	A policy change is already in progress.
`LYNXSK_MSP_UNKNOWN_POLICY`	The policy ID is invalid.

FLEX_SCHD_GIVE_ALL - Donate All Time

Synopsis

```
retval = VMCALL(HVCALL_FLEX_SCHD_GIVE_ALL, subjectid);
```

Description

The requesting VCPU donates its scheduled time to the specified subject indefinitely.

Arguments

Argument Name	Description
`uint32_t subjectid`	The identifier of the recipient subject.

Permissions

```
<subject …>…<subjectflow sname=subjectname … flexdonateperm="ALLOW" …/>…</subject>
```

Notes

- The target subject must be scheduled to run on the same physical processor as the requesting subject. If the target subject is an SMP subject (i.e. contains more than one VCPU), the VCPU running on the same physical processor is the recipient of the donated time. If the requesting subject is an SMP subject, only the time slice of the requesting VCPU is donated.
- The target subject may further donate this time with appropriate authorization, or yield it back to the original owner using the `HVCALL_FLEX_RETURN` hypercall.
- If the time is further donated to another subject, it will return all the way to the original owner when any of the return conditions used in the donation chain occurs.
- The donated time slice includes all of the minor frames in the current scheduling policy owned by the original owner of the current minor frame.

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded. Note that the hypercall returns success only when the calling VCPU gets some CPU time again.
`LYNXSK_PDE_PERMISSION_DENIED`	The subject ID is invalid or the calling subject has no permission to perform the operation.
`LYNXSK_INVALID_ARG`	The target subject does not have a VCPU running on the same physical processor as the caller of this hypercall.

FLEX_SCHD_GIVE_UNTIL_EVENT - Donate Until an Asynchronous Event

Synopsis

```
retval = VMCALL(HVCALL_FLEX_SCHD_GIVE_UNTIL_EVENT, subjectid);
```

Description

The requesting VCPU donates its time to the specified subject until an asynchronous event. Time is automatically returned to the original owner as soon as an asynchronous event is detected designated for the donor VCPU (not the time recipient VCPU). The asynchronous events include:

- An interrupt, including timer expirations
- A subject state change
- A VCPU state change
- Certain internal LynxSecure® events

If there is an asynchronous event already pending against the calling VCPU at the time this hypercall is issued, the hypercall returns immediately with a success code, but without actually donating time. This includes the case where a pending interrupt delivery is disabled by a CPU interrupt masking flag. This does not, however, include the case where the pending interrupt is blocked by elevated interrupt priority or masked in the virtual interrupt controller.

Arguments

Argument Name	Description
`uint32_t subjectid`	The identifier of the recipient subject.

Permissions

`<subject …>…<subjectflow sname=subjectname … flexdonateperm="ALLOW" …/>…</subject>`

Notes

- The target subject must be scheduled to run on the same physical processor as the requesting subject. If the target subject is an SMP subject (i.e. contains more than one VCPU), the VCPU running on the same physical processor is the recipient of the donated time. If the requesting subject is an SMP subject, only the time slice of the requesting VCPU is donated.
- Unlike the indefinite donation and donation until the next SCT, only the time slice owner may donate its time until an asynchronous event. In other words, nested donation until an asynchronous event is not allowed (regardless of how the time was donated to the subject making this hypercall).
- The target subject may further donate this time with appropriate authorization, or yield it back to the original owner using the `HVCALL_FLEX_RETURN` hypercall.
- If the time is further donated to another subject, it will return all the way to the original owner when any of the return conditions used in the donation chain occurs.
- The donated time slice includes all of the minor frames in the current scheduling policy owned by the original owner of the current minor frame.

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded. Note that the hypercall returns success only when the calling VCPU gets some CPU time again, unless an asynchronous event was already pending at the time this hypercall was issued.
`LYNXSK_FAILURE`	A nested donation until an asynchronous event has been attempted.

Return Value	Description
LYNXSK_PDE_PERMISSION_DENIED	The subject ID is invalid or the calling subject has no permission to perform the operation.
LYNXSK_INVALID_ARG	The target subject does not have a VCPU running on the same physical processor as the caller of this hypercall.

FLEX_SCHD_GIVE_UNTIL_SCT - Donate Until System Clock Tick

Synopsis

```
retval = VMCALL(HVCALL_FLEX_SCHD_GIVE_UNTIL_SCT, subjectid);
```

Description

The requesting VCPU donates its scheduled time to the specified subject until the next system clock tick interrupt, when the time slice is automatically yielded back to the original owner.

Arguments

Argument Name	Description
uint32_t subjectid	The identifier of the recipient subject.

Permissions

```
<subject …>…<subjectflow sname=subjectname … flexdonateperm="ALLOW" …/>…</subject>
```

Notes

- The target subject must be scheduled to run on the same physical processor as the requesting subject. If the target subject is an SMP subject (i.e. contains more than one VCPU), the VCPU running on the same physical processor is the recipient of the donated time. If the requesting subject is an SMP subject, only the time slice of the requesting VCPU is donated.
- The target subject may further donate this time with appropriate authorization, or yield it back to the original owner using the `HVCALL_FLEX_RETURN` hypercall.
- If the time is further donated to another subject, it will return all the way to the original owner when any of the return conditions used in the donation chain occurs.
- The donated time slice includes all of the minor frames in the current scheduling policy owned by the original owner of the current minor frame.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded. Note that the hypercall returns success only when the calling VCPU gets some CPU time again.
LYNXSK_PDE_PERMISSION_DENIED	The subject ID is invalid or the calling subject has no permission to perform the operation.
LYNXSK_INVALID_ARG	The target subject does not have a VCPU running on the same physical processor as the caller of this hypercall.

FLEX_SCHD_RETURN - Yield Donated Time

Synopsis

```
retval = VMCALL(HVCALL_FLEX_SCHD_RETURN);
```

Description

The requesting subject returns donated schedule time back to the subject scheduled for the current minor frame in the configured scheduling policy.

Arguments

None

Permissions

```
<subject sname=subjectname … flexreturnperm="ALLOW" …>…</subject>
```

Notes

In order to use this hypercall, the subject must have the permission to yield its time back to the original owner in the configuration vector. If the subject is not authorized to do so, this hypercall has no effect. A subject with no permission to use this hypercall could donate the time to itself until the next SCT interrupt, which will effectively perform a yield to the original time owner at the SCT interrupt.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.

GET_ROPAGE_INFO - Obtain RO Page Information

Synopsis

```
ropage_info_t info;
uint64_t p_info = (uint64_t)&info;
retval = VMCALL(HVCALL_GET_ROPAGE_INFO, p_info);
```

Description

Saves RO page information to a structure of type `ropage_info_t`. The info structure contains the following fields:

Field	Description
uint64_t paddr	The guest physical address of the RO page.
uint64_t initial_addr	The address where the RO page is mapped when subject code starts executing for the first time (a virtual address for paravirtualized subjects, a physical address equal to `paddr` for fully virtualized subjects).
uint64_t size	RO page size in bytes

Arguments

Argument Name	Description
`uint64_t p_info`	The GVA of the info structure structure of the type `ropage_info_t`.

Permissions

None

Notes

None

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded.
`LYNXSK_ADDRESS_FAULT`	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
`LYNXSK_BAD_MEMORY`	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
`LYNXSK_IO_ERROR`	The memory subsystem has reported a hardware error.

GET_SCHD_POLICY_V - Read Active Scheduling Policy Identifier

Synopsis

```
retval = VMCALL(HVCALL_GET_SCHD_POLICY_V, p_policy_id);
```

Description

Returns the identifier of the currently active scheduling policy.

Arguments

Argument Name	Description
`uint64_t p_policy_id`	GVA of a `uint32_t` variable in which to store the current scheduling policy identifer.

Permissions

```
<subject … readschedpolicyperm="ALLOW" …/>
```

Notes

None

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded.
`LYNXSK_ADDRESS_FAULT`	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.

Return Value	Description
LYNXSK_BAD_MEMORY	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
LYNXSK_IO_ERROR	The memory subsystem has reported a hardware error.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.

INTR_EOI - End of Interrupt

Synopsis

```
retval = VMCALL(HVCALL_INTR_EOI);
```

Description

Invoked by a paravirtualized subject to indicate that the calling VCPU has finished processing an interrupt. This call has to be made for all types of external interrupts in a subject, including synthetic ones and those coming from external devices.

Arguments

None

Permissions

None

Notes

Marks the highest priority interrupt currently being serviced by the calling VCPU as no longer being serviced. If there is no interrupt being serviced, this hypercall returns an error.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_FAILURE	The hypercall has failed (no in-service interrupt or not called by a paravirtualized subject).

MSG_RECV_V - Read Message from Message Buffer

Synopsis

```
retval = VMCALL(HVCALL_MSG_RECV_V, msgbufid, msgbuf);
```

Description

Reads a 64-byte message from the identified message channel.

Arguments

Argument Name	Description
uint32_t msgbufid	Object identifier of the message channel.

Argument Name	Description
`uint64_t msgbuf`	GVA of a 64-byte buffer to receive a message.

Permissions

`<subject…>…<messagebufferflow msgbufname=msgbufname readperm="ALLOW"/>…</subject>`

Notes

None

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded.
`LYNXSK_ADDRESS_FAULT`	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
`LYNXSK_BAD_MEMORY`	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
`LYNXSK_IO_ERROR`	The memory subsystem has reported a hardware error.
`LYNXSK_INVALID_ARG`	Bad message channel ID, or the calling subject is not on the receiving end of the channel according to the configuration.
`LYNXSK_FAILURE`	The channel is empty (no message to read).

MSG_SEND_V - Write Message to Message Buffer

Synopsis

```
retval = VMCALL(HVCALL_MSG_SEND_V, msgbufid, msgbuf);
```

Description

Writes a 64-byte message to the identified message channel. May cause a synthetic interrupt to be injected into the other subject (if configured in the flow).

Arguments

Argument Name	Description
`uint32_t msgbufid`	Object identifier of the message channel.
`uint64_t msgbuf`	GVA of a 64-byte buffer that contains the message to be sent.

Permissions

`<subject…>…<messagebufferflow msgbufname=msgbufname writeperm="ALLOW"/>…</subject>`

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_ADDRESS_FAULT	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
LYNXSK_BAD_MEMORY	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
LYNXSK_IO_ERROR	The memory subsystem has reported a hardware error.
LYNXSK_INVALID_ARG	Bad message channel ID, or the calling subject is not on the sending end of the channel according to the configuration.
LYNXSK_FAILURE	The channel is full.

RETRIEVE_AUDIT_RECORD_V - Read a Record from the Audit Buffer

Synopsis

```
lynxsk_stored_audit_record_t auditbuf;
uint64_t p_auditbuf = (uint64_t)&auditbuf;
retval = VMCALL(HVCALL_RETRIEVE_AUDIT_RECORD_V, p_auditbuf);
```

Description

Retrieve an audit record from the audit buffer log. The audit record structure is of the type `lynxsk_stored_audit_record_t` and contains the following fields:

Field	Description
char comment_field[AUDIT_COMMENT_FIELD_SIZE];	Stores the data for a subject-generated Audit Record or stores extra information for Audit Records generated internally by LynxSecure
ls_ername_id_t initiator_id	Identifier of an exported resource or LynxSecure object that initiates an action which generates an Audit Record.
ls_ername_id_t recipient_id	Identifier of an exported resource or LynxSecure object that is the target of an action which generates an Audit Record.
uint32_t audit_event_type	The specific event type that caused the Audit Record to be generated. See Appendix B for the list of the audit event types.
abs_time_t timestamp	The Wall time timestamp of the audit event.

Arguments

Argument Name	Description
uint64_t p_auditbuf	GVA of the buffer to receive an audit event. The type of the buffer is `lynxsk_stored_audit_record_t`.

Permissions

```
<subject …>…<auditflow auditname=auditname readperm="ALLOW"…/>…</subject>
```

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_ADDRESS_FAULT	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
LYNXSK_BAD_MEMORY	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
LYNXSK_IO_ERROR	The memory subsystem has reported a hardware error.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.
LYNXSK_AUDIT_EMPTY_AUDIT_BUFFER	The audit buffer is empty.

RETRIEVE_OVERFLOW_AUDIT_RECORD_V - Read an Audit Buffer Overflow Record

Synopsis

```
lynxsk_overflow_audit_record_t auditbuf;
uint64_t p_auditbuf = (uint64_t)&auditbuf;
retval = VMCALL(HVCALL_RETRIEVE_OVERFLOW_AUDIT_RECORD_V, p_auditbuf);
```

Description

Retrieve an overflow audit record from the audit log.

Arguments

Argument Name	Description
uint64_t p_auditbuf	GVA of the buffer to receive an audit event. The type of the buffer is lynxsk_overflow_audit_record_t.

Permissions

```
<subject …>…<auditflow auditname=auditname readperm="ALLOW"…/>…</subject>
```

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_ADDRESS_FAULT	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
LYNXSK_BAD_MEMORY	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
LYNXSK_IO_ERROR	The memory subsystem has reported a hardware error.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.

ROUTE_INTR - Set Interrupt Routing

Synopsis

```
retval = VMCALL(HVCALL_ROUTE_INTR, vector, p_cpu_mask, mask_size);
```

Description

Specify how an interrupt vector is routed to VCPUs in the calling subject. Whenever an interrupt to this vector is injected into the subject, the interrupt is delivered to target VCPUs listed in the specified bit mask.

In the bit mask, a set bit means that the interrupt is delivered to the VCPU ID matching the number of the bit. More than one target VCPU may be specified at one time for one vector.

If the interrupt vector is generated by an external device, at least one VCPU must be specified as the target VCPU. This is a hardware limitation. On the other hand, if the interrupt vector is synthetic, the mask may be empty resulting in the interrupt being dropped rather than delivered to any VCPU. The system clock tick interrupt is considered a synthetic interrupt for this purpose.

Initially, all interrupt vectors in the subject are routed to VCPU 0 only.

Arguments

Argument Name	Description
`uint32_t vector`	The interrupt vector (not the IRQ number) to set the routing for.
`uint64_t p_cpu_mask`	GVA of the target VCPU bit mask in memory.
`uint64_t mask_size`	The size of the CPU bit mask in bytes.

Permissions

None

Notes

None

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded.
`LYNXSK_INVALID_ARG`	The interrupt vector is invalid, or non-existent VCPUs are listed in the mask, or the vector is generated by an external device and the mask is empty.
`LYNXSK_ADDRESS_FAULT`	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
`LYNXSK_BAD_MEMORY`	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
`LYNXSK_IO_ERROR`	The memory subsystem has reported a hardware error.

SEND_IPI - Send an Inter-Processor Interrupt

Synopsis

```
retval = VMCALL(HVCALL_SEND_IPI, vcpuid, vector);
```

Description

Send an inter-processor interrupt to a VCPU in the same subject. The interrupt with the specified vector is injected into the target VCPU.

This call can also halt and start VCPUs. If the special value `LYNXSK_SEND_IPI_INIT` is specified for the vector, the target VCPU is reset and halted. If the special value `LYNXSK_SEND_IPI_START` is specified for the vector, the target VCPU starts execution at the subject start addess specified in the configuration. For paravirtualized subjects, the newly started VCPU will use the subject's initial page table.

When a subject is started or restarted, all VCPUs in the subject except VCPU 0 are halted and must be explicitly started using this hypercall.

VCPU IDs range from 0 (the boot VCPU) to N-1, where N is the number of VCPUs in the subject. It can be found in the RO page field `cpus_num`.

A VCPU may find out its VCPU ID using the CPUID instruction (it is reported as the initial APIC ID of the VCPU).

Fully virtualized subjects must use the hardware mechanisms provided by the virtual platform instead of this hypercall to send IPIs.

Arguments

Argument Name	Description
`uint32_t vcpuid`	The VCPU ID of the target VCPU in the calling subject.
`uint32_t vector`	The interrupt vector (not the IRQ number) to inject into the target VCPU. Special values may be specified: LYNXSK_SEND_IPI_INIT - reset and halt the target VCPU LYNXSK_SEND_IPI_START - start the target VCPU

Permissions

None

Notes

None

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded.
`LYNXSK_FAILURE`	The hypercall was invoked from a fully virtualized subject.
`LYNXSK_INVALID_ARG`	The VCPU ID or the vector is invalid.

SESM_GET_STATE_V - Get Subject Status

Synopsis

```
retval = VMCALL(HVCALL_SESM_GET_STATE_V, subjectid, p_state);
```

Description

Retrieve the state of a subject. The state is one of: `SUBJECT_RUNNING`, `SUBJECT_STOPPED`, `SUBJECT_SUSPENDED`.

Arguments

Argument Name	Description
`uint32_t subjectid`	Object identifier of the subject to obtain the state of.
`uint64_t p_state`	GVA of a `uint32_t` location where to store the state value.

Permissions

```
<subject …>…<subjectflow sname=subjectname … statusperm="ALLOW" …/>…</subject>
```

Notes

None

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded.
`LYNXSK_ADDRESS_FAULT`	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
`LYNXSK_BAD_MEMORY`	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
`LYNXSK_IO_ERROR`	The memory subsystem has reported a hardware error.
`LYNXSK_PDE_PERMISSION_DENIED`	The subject ID is invalid, or the calling subject has no permission to perform the operation on the given subject.

SESM_RESTART_SUBJECT - Restart Subject

Synopsis

```
retval = VMCALL(HVCALL_SESM_RESTART_SUBJECT, subjectid);
```

Description

Stops and starts the target subject; makes an audit entry of the state change. The state change will be effective upon next major frame and time window for the target subject.

Arguments

Argument Name	Description
`uint32_t subjectid`	Object identifier of the subject to restart.

Permissions

```
<subject …>…<subjectflow sname=subjectname … restartperm="ALLOW" …/>…</subject>
```

Notes

- The subject may be in any state.

- The subject's initial `PROGRAM` memory region is reinitialized, but other memory regions assigned to the subject are unchanged.
- The state change occurs after all VCPUs in the target subject react to the request.
- If a subject state change is requested before any previous request is completed, the request is queued. LynxSecure queues up to one request; any subsequent requests override the queued request.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_SESM_STATE_CHANGE_FAILED	The subject state change requested is not valid.
LYNXSK_PDE_PERMISSION_DENIED	The subject ID is invalid, or the calling subject has no permission to perform the operation on the given subject.

SESM_RESUME_SUBJECT - Resume Suspended Subject

Synopsis

```
retval = VMCALL(HVCALL_SESM_RESUME_SUBJECT, subjectid);
```

Description

Sets the target subject to the running state from suspended state and makes an audit entry of the state change. The state change will be effective upon next major frame and time window for the target subject.

Arguments

Argument Name	Description
uint32_t subjectid	Object identifier of the subject to resume.

Permissions

```
<subject …>…<subjectflow sname=subjectname … resumeperm="ALLOW" …/>…</subject>
```

Notes

- The subject must be in the suspended state.
- The state change occurs after all VCPUs in the target subject react to the request.
- If a subject state change is requested before any previous request is completed, the request is queued. LynxSecure queues up to one request; any subsequent requests override the queued request.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_SESM_STATE_CHANGE_FAILED	The subject state change requested is not valid.
LYNXSK_PDE_PERMISSION_DENIED	The subject ID is invalid, or the calling subject has no permission to perform the operation on the given subject.

SESM_START_SUBJECT - Start Stopped Subject

Synopsis

```
retval = VMCALL(HVCALL_SESM_START_SUBJECT, subjectid);
```

Description

Sets the target subject to running state from stopped state and make an audit entry of the state change. The state change will be effective upon next major frame and time window for the target subject.

Arguments

Argument Name	Description
`uint32_t subjectid`	Object identifier of the subject to start.

Permissions

```
<subject …>…<subjectflow sname=subjectname … startperm="ALLOW" …/>…</subject>
```

Notes

- The subject must be in the stopped state.
- The subject's initial `PROGRAM` memory region is reinitialized, but other memory regions assigned to the subject are unchanged.
- The state change occurs after all VCPUs in the target subject react to the request.
- If a subject state change is requested before any previous request is completed, the request is queued. LynxSecure queues up to one request; any subsequent requests override the queued request.

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded.
`LYNXSK_SESM_STATE_CHANGE_FAILED`	The subject state change requested is not valid.
`LYNXSK_PDE_PERMISSION_DENIED`	The subject ID is invalid, or the calling subject has no permission to perform the operation on the given subject.

SESM_STOP_SUBJECT - Stop Subject

Synopsis

```
retval = VMCALL(HVCALL_SESM_STOP_SUBJECT, subjectid);
```

Description

Sets the target subject to the stopped state and makes an audit entry of the state change. The state change will be effective upon next major frame and time window for the target subject.

Arguments

Argument Name	Description
`uint32_t subjectid`	Object identifier of the subject to stop.

Permissions

```
<subject …>…<subjectflow sname=subjectname … stopperm="ALLOW" …/>…</subject>
```

Notes

- Once stopped, the subject may only be started from its initial state.
- The state change occurs after all VCPUs in the target subject react to the request.
- If a subject state change is requested before any previous request is completed, the request is queued. LynxSecure queues up to one request; any subsequent requests override the queued request.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_SESM_STATE_CHANGE_FAILED	The subject state change requested is not valid.
LYNXSK_PDE_PERMISSION_DENIED	The subject ID is invalid, or the calling subject has no permission to perform the operation on the given subject.

SESM_SUSPEND_SUBJECT - Suspend Subject

Synopsis

```
retval = VMCALL(HVCALL_SESM_SUSPEND_SUBJECT, subjectid);
```

Description

Sets the target subject to the suspended state and makes an audit entry of the state change. The state change will be effective upon next major frame and time window for the target subject.

When a subject is suspended, all its VCPUs are frozen. The subject can be resumed with the HVCALL_SESM_RESUME_SUBJECT hypercall.

Arguments

Argument Name	Description
uint32_t subjectid	Object identifier of the subject to suspend.

Permissions

```
<subject …>…<subjectflow sname=subjectname … suspendperm="ALLOW" …/>…</subject>
```

Notes

- The state change occurs after all VCPUs in the target subject react to the request.
- All interrupts to the suspended subject are queued for delivery. Once the subject resumes, all the interrupts are delivered.
- If a subject state change is requested before any previous request is completed, the request is queued. LynxSecure queues up to one request; any subsequent requests override the queued request.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_SESM_STATE_CHANGE_FAILED	The subject state change requested is not valid.
LYNXSK_PDE_PERMISSION_DENIED	The subject ID is invalid, or the calling subject has no permission to perform the operation on the given subject.

SET_HRTIMER - Schedule the High-Resolution Timer Expiration

Synopsis

```
retval = VMCALL(HVCALL_SET_HRTIMER, seconds, nanoseconds, type);
```

Description

Schedule an expiration of the high-resolution timer. Once the timer expires, an interrupt is injected into the calling VCPU. The interrupt vector can be set with the `HVCALL_SET_HRTIMER_VEC` hypercall. Each VCPU in each subject has one independent high-resolution timer.

The expiration time is specified relative to the current Monotonic time. The interrupt will be delivered as soon as possible after the specified expiration time. The delivery may be delayed by subject schedule and other activity in the system.

This hypercall is only available on the x86 platform and only to paravirtualized subjects. In all other cases, subjects must use platform high-resolution timers instead.

Arguments

Argument Name	Description
uint64_t seconds	Expiration seconds.
uint64_t nanoseconds	Expiration nanoseconds.
uint32_t type	0 to create a relative timer: the expiration is the current Monotonic time plus the specified seconds/nanoseconds. All other values of this argument are invalid and reserved for future expansion.

Permissions

None

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_FAILURE	The hypercall was not invoked by a paravirtualized subject.
LYNXSK_INVALID_ARG	The expiration time value is invalid: either the nanoseconds are one billion or above, or the seconds are greater than $((2^{63}-1)/10^{9}-1)$.

SET_HRTIMER_VEC - Set the High-Resolution Timer Vector

Synopsis

```
retval = VMCALL(HVCALL_SET_HRTIMER_VEC, vector);
```

Description

Specify the interrupt vector delivered on the high-resolution timer expiration in the current subject. This affects all VCPUs in the subject.

If the vector is equal to `LYNXSK_SET_HRTIMER_VEC_NONE`, no interrupt is delivered on the high-resolution timer expiration. This is the initial configuration of each subject.

This hypercall is only available on the x86 platform and only to paravirtualized subjects. In all other cases, subjects must use platform high-resolution timers instead.

Arguments

Argument Name	Description
`uint32_t vector`	The interrupt vector (not the IRQ number) to deliver on the timer expiration. Special values may be specified: `LYNXSK_SET_HRTIMER_VEC_NONE` - no interrupt is delivered.

Permissions

None

Notes

None

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded.
`LYNXSK_FAILURE`	The hypercall was not invoked by a paravirtualized subject.
`LYNXSK_INVALID_ARG`	The vector value is invalid.

SKDB_ENTER - Enter SKDB

Synopsis

```
retval = VMCALL(HVCALL_SKDB_ENTER);
```

Description

If the SKDB module is enabled in the HCV, this hypercall will force an entry into SKDB. SKDB will use the consoles configured in the hypervisor for output. At this time, input is only supported from the serial console.

Arguments

None

Permissions

```
<subject … skdbperm="ALLOW" …/>
```

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	Hypercall was successful (hypercall will return when SKDB command loop terminates).
LYNXSK_PDE_PERMISSION_DENIED	The calling subject does not have a permission to request SKDB entry.

SKDB_EXECUTE - Execute SKDB Command(s)

Synopsis

```
skdb_exec_t ex;
uint64_t p_ex = (uint64_t)&ex;
retval = VMCALL(HVCALL_SKDB_EXECUTE, p_ex);
```

Description

If the SKDB module is enabled in the HCV, this hypercall allows one to execute SKDB commands without entering an interactive command loop. Instead, the command (or commands) are placed into a memory buffer, and another memory buffer is supplied to store the commands' output. The hypervisor will enter the SKDB as usual (stopping other CPUs), execute the supplied commands and resume normal execution. This hypercall is primarily intended for execution of non-invasive SKDB commands.

The commands are supplied in the input buffer, each command must be terminated by the newline character. The input buffer length should not include the terminating NUL character, if any. Upon successful return from the hypercall, the output buffer will store the output of the executed commands. The output buffer is not NUL-terminated either. The caller must zero the output buffer prior to this hypercall and pass the length one byte less than the size of the buffer (to ensure the last terminating NUL is preserved).

The structure passed to this hypercall contains the following fields:

Field	Description
uint64_t input_addr	The GVA of the input buffer.
uint64_t input_size	The size of the input buffer in bytes.
uint64_t output_addr	The GVA of the output buffer.
uint64_t output_size	The size of the output buffer in bytes.

Arguments

Argument Name	Description
uint64_t p_ex	The GVA of the structure with the input/output buffers of the type skdb_exec_t.

Permissions

```
<subject … skdbperm="ALLOW" …/>
```

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	Hypercall was successful (hypercall will return when SKDB command loop terminates).
LYNXSK_PDE_PERMISSION_DENIED	The calling subject does not have a permission to request SKDB entry.
LYNXSK_ADDRESS_FAULT	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
LYNXSK_BAD_MEMORY	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
LYNXSK_IO_ERROR	The memory subsystem has reported a hardware error.

SSM_GET_STATE_V - Get System State

Synopsis

```
retval = VMCALL(HVCALL_SSM_GET_STATE_V, p_state);
```

Description

Return the state of the LynxSecure Separation Kernel/Hypervisor in the location pointed to by the argument. The state is one of the following:

- SSM_INITIAL_STATE
- SSM_STARTUP
- SSM_VALIDATION
- SSM_OPERATION
- SSM_MAINTENANCE_INSECURE
- SSM_MAINTENANCE_SECURE
- SSM_INITIATED_BIT
- SSM_SHUTDOWN
- SSM_RESTART

Arguments

Argument Name	Description
uint64_t p_state	GVA of a uint32_t variable to contain the current state.

Permissions

None

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_ADDRESS_FAULT	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
LYNXSK_BAD_MEMORY	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
LYNXSK_IO_ERROR	The memory subsystem has reported a hardware error.
LYNXSK_SSM_PERMISSION_DENIED	The hypercall is not permitted in the current system state.

SSM_SET_INITIATED_BIT - Initiate Built-In Test

Synopsis

```
retval = VMCALL(HVCALL_SSM_SET_INITIATED_BIT);
```

Description

Initiates Built-In Test (BIT).

Arguments

None

Permissions

```
<subject … bitperm="ALLOW" …/>
```

Notes

Initiated BIT (IBIT) is run in a separate scheduling policy where other subjects do not run.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.
LYNXSK_SSM_PERMISSION_DENIED	The hypercall is not permitted in the current system state.
LYNXSK_FAILURE	The requested state transition is not permitted in the current system state.

SSM_SET_LAST_STATE - SSM Set Last State

Synopsis

```
retval = VMCALL(HVCALL_SSM_SET_LAST_STATE);
```

Description

If permitted, transitions the system to the last state entered before the current state.

Arguments

None

Permissions

None

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_SSM_PERMISSION_DENIED	The hypercall is not permitted in the current system state.
LYNXSK_FAILURE	The requested state transition is not permitted in the current system state.

SSM_SET_MAINTENANCE_INSECURE - Initiate Transition to Maintenance Insecure Mode

Synopsis

```
retval = VMCALL(HVCALL_SSM_SET_MAINTENANCE_INSECURE);
```

Description

If permitted, transitions the system to the Maintenance Insecure state.

LynxSecure audits this event, suspends all non critical subjects. If no maintenance scheduling policy is specified or if there was error transitioning to maintenance scheduling policy a shutdown occurs.

Arguments

None

Permissions

```
<subject … maintperm="ALLOW" …/>
```

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.
LYNXSK_SSM_PERMISSION_DENIED	The hypercall is not permitted in the current system state.
LYNXSK_FAILURE	The requested state transition is not permitted in the current system state.

SSM_SET_MAINTENANCE_SECURE - Initiate Transition to Maintenance Mode

Synopsis

```
retval = VMCALL(HVCALL_SSM_SET_MAINTENANCE_SECURE);
```

Description

Initiates a transition to Maintenance Secure mode.

Arguments

None

Permissions

```
<subject … maintperm="ALLOW" …/>
```

Notes

- If no maintenance mode scheduling policy is defined, this hypercall is equivalent to `HVCALL_SSM_SET_SHUTDOWN`.
- The maintenance mode scheduling policy becomes effective at the end of current major frame.
- All subjects are immediately suspended with the exception of those that also appear in the maintenance mode scheduling policy. This ensures that safety-critical subjects continue to run without interruption.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.
LYNXSK_SSM_PERMISSION_DENIED	The hypercall is not permitted in the current system state.
LYNXSK_FAILURE	The requested state transition is not permitted in the current system state.

SSM_SET_OPERATION - Set System State to Operation

Synopsis

```
retval = VMCALL(HVCALL_SSM_SET_OPERATION);
```

Description

Initiates a transition to Operation mode.

Arguments

None

Permissions

None

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_SSM_PERMISSION_DENIED	The hypercall is not permitted in the current system state.
LYNXSK_FAILURE	The requested state transition is not permitted in the current system state.

SSM_SET_RESTART - Initiate System Restart

Synopsis

```
retval = VMCALL(HVCALL_SSM_SET_RESTART);
```

Description

Initiates a restart of the system.

Arguments

None

Permissions

```
<subject … restartsysperm="ALLOW" …/>
```

Notes

All subjects are immediately stopped; they are not gracefully shutdown.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.
LYNXSK_SSM_PERMISSION_DENIED	The hypercall is not permitted in the current system state.
LYNXSK_FAILURE	The requested state transition is not permitted in the current system state.

SSM_SET_SHUTDOWN - Shut Down the System

Synopsis

```
retval = VMCALL(HVCALL_SSM_SET_SHUTDOWN);
```

Description

Initiates a shutdown of the system.

Arguments

None

Permissions

`<subject … haltsysperm="ALLOW" …/>`

Notes

All subjects are immediately stopped; they are not gracefully shutdown.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.
LYNXSK_SSM_PERMISSION_DENIED	The hypercall is not permitted in the current system state.
LYNXSK_FAILURE	The requested state transition is not permitted in the current system state.

STORE_AUDIT_RECORD_V - Store Audit Record

Synopsis

```
retval = VMCALL(HVCALL_STORE_AUDIT_RECORD_V, p_auditbuf);
```

Description

Stores audit events in the LynxSecure audit log.

Arguments

Argument Name	Description
uint64_t p_auditbuf	GVA of the buffer that contains the comment part of the audit structure. The buffer length is AUDIT_COMMENT_FIELD_SIZE bytes.

Permissions

`<subject …>…<auditflow auditname=auditname writeperm="ALLOW"/>…</subject>`

Notes

The buffer provided via this hypercall contains only the part of the audit record to be stored in the `comment_field` member of `lynxsk_stored_audit_record_t` structure. The buffer length is `AUDIT_COMMENT_FIELD_SIZE` bytes. The buffer typically contains a string. The rest of of the fields of the `lynxsk_stored_audit_record_t` structure is filled by LynxSecure automatically.

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.

Return Value	Description
LYNXSK_ADDRESS_FAULT	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
LYNXSK_BAD_MEMORY	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
LYNXSK_IO_ERROR	The memory subsystem has reported a hardware error.
LYNXSK_AUDIT_FULL_AUDIT_BUFFER	The audit buffer is full.
LYNXSK_AUDIT_FULL_BUFFER_ACTION_ ERROR	The audit buffer is full and the system has failed to perform the action associated with the buffer overflow event.

STROBE_WATCHDOG - Strobe Subject Watchdog

Synopsis

```
retval = VMCALL(HVCALL_STROBE_WATCHDOG);
```

Description

This hypercall must be invoked periodically by subjects which have configured software watchdog. If a subject does not make this hypercall within the configured time frame, the configured action will be taken.

Refer to section “<subject> — Subject” in *LynxSecure 6.3.0 Advanced Configuration Guide* for the information on configuring the subject watchdog.

Arguments

None

Permissions

```
<subject … on_watchdog="…" watchdog_timeout="…" …/>
```

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.

SUBJECT_LOG - Print a Message from a Subject

Synopsis

```
retval = VMCALL(HVCALL_SUBJECT_LOG, p_str);
```

Description

This hypercall prints the string passed as the argument to the configured hypervisor's consoles. The string is limited to `SUBJECT_LOG_SIZE` characters. The printed message is prefixed with the subject's name.

This hypercall can be useful for early debug of paravirtualized subjects or LynxSecure applications (LSAs).

Arguments

Argument Name	Description
`uint64_t p_str`	The GVA of the string to be printed.

Permissions

None

Notes

Due to timing constraints and potential interference with devices assigned to other subjects, this hypercall is not enabled in the default configuration of LynxSecure. To enable this hypercall, relink the SKH binary as described in section "Enabling Additional Modules" in *LynxSecure 6.3.0 Advanced Configuration Guide*.

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded.
`LYNXSK_ADDRESS_FAULT`	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
`LYNXSK_BAD_MEMORY`	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
`LYNXSK_IO_ERROR`	The memory subsystem has reported a hardware error.

SYNTH_INTR - Synthetic Interrupt

Synopsis

```
retval = VMCALL(HVCALL_SYNTH_INTR, irq, subject_id);
```

Description

Inject a synthetic interrupt with the specified IRQ number into the target subject.

Note that the IRQ number is not the same as the interrupt vector. The resulting interrupt vector injected into the target subject equals the IRQ number plus that subject's IRQ base specified in the configuration (if not specified, defaults to 32).

Arguments

Argument Name	Description
`int irq`	The interrupt IRQ number to inject.
`uint32_t subject_id`	Target subject ID to receive the interrupt.

Permissions

```
<subject …>…<subjectflow sname=subjectname…>…<injectintperm irq="irq" perm="ALLOW"/>…</
subjectflow>…</subject>
```

Notes

- If the target subject is in the suspended state, the interrupts are queued.

- Interrupts are ignored if the target state is "stopped".

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_PDE_PERMISSION_DENIED	The subject ID is invalid.
LYNXSK_FAILURE	The calling subject is not permitted to inject the interrupt in the configuration vector.

TCAL_GET_V - Get the Wall Time Calibration Value

Synopsis

```
retval = VMCALL(HVCALL_TCAL_GET_V, p_tcal);
```

Description

Returns the Wall time timebase calibration value. See the `HVCALL_TCAL_SET` hypercall description below (page 50) for details on how to interpret the value.

Arguments

Argument Name	Description
uint64_t p_tcal	GVA of a `uint64_t` variable to store the timebase calibration value in.

Permissions

```
<subject …>…<absoluteclockflow readcalibrationperm="ALLOW"/>…</subject>
```

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_ADDRESS_FAULT	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
LYNXSK_BAD_MEMORY	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
LYNXSK_IO_ERROR	The memory subsystem has reported a hardware error.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.

TCAL_SET - Set the Wall Time Calibration Value

Synopsis

```
retval = VMCALL(HVCALL_TCAL_SET, tcal);
```

Description

Set the Wall time timebase calibration value. The value is the amount of Wall time it takes for one increment of the system time base register (the Time Stamp Counter for the x86 platform, or the Counter-timer Physical Count for the ARM platform). The upper `64-TCAL_FRAC_BITS` bits of this value is the number of whole nanoseconds. The low `TCAL_FRAC_BITS` is the fractional nanosecond part. In other words, `tcal` = `nanoseconds` * $2^{\mathtt{TCAL\_FRAC\_BITS}}$. The timebase calibration value affects the rate of the Wall time and is intended for smooth Wall time adjustments. The timebase calibration value does not affect the Monotonic time.

Arguments

Argument Name	Description
`uint64_t tcal`	The Wall time calibration value to set.

Permissions

```
<subject …>…<absoluteclockflow writecalibrationperm="ALLOW"/>…</subject>
```

Notes

This operation changes the speed of the Wall time flow.

Return Values

Return Value	Description
`LYNXSK_SUCCESS`	The hypercall has succeeded.
`LYNXSK_PDE_PERMISSION_DENIED`	The calling subject has no permission to perform the operation.
`LYNXSK_TIMEKEEPING_BAD_CAL`	The calibration value has failed a sanity check.

TIME_GET_MONOTONIC - Get Absolute Clock Monotonic Time

Synopsis

```
abs_time_t abstime;
uint64_t p_abstime = (uint64_t)&abstime;
retval = VMCALL(HVCALL_TIME_GET_MONOTONIC, p_abstime);
```

Description

Reads the current Monotonic time which is the time in seconds and nanoseconds since LynxSecure startup. The Monotonic time is written to the location pointed to by the argument.

Arguments

Argument Name	Description
`uint64_t p_abstime`	GVA of an `abs_time_t` variable to write the time to.

Permissions

```
<absoluteclockflow absclockname=clockname readabsclockperm="ALLOW"/>
```

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_ADDRESS_FAULT	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
LYNXSK_BAD_MEMORY	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
LYNXSK_IO_ERROR	The memory subsystem has reported a hardware error.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.

TIME_GET_V - Get Absolute Clock Wall Time

Synopsis

```
abs_time_t abstime;
uint64_t p_abstime = (uint64_t)&abstime;
retval = VMCALL(HVCALL_TIME_GET_V, p_abstime);
```

Description

Reads the current Wall time which is the time in seconds and nanoseconds since the Epoch (Jan 1 1970 12:00AM). The Wall time is written to the location pointed to by the argument.

Arguments

Argument Name	Description
uint64_t p_abstime	GVA of an abs_time_t variable to write the time to.

Permissions

```
<absoluteclockflow absclockname=clockname readabsclockperm="ALLOW"/>
```

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_ADDRESS_FAULT	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
LYNXSK_BAD_MEMORY	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
LYNXSK_IO_ERROR	The memory subsystem has reported a hardware error.
LYNXSK_PDE_PERMISSION_DENIED	The calling subject has no permission to perform the operation.

TIME_LEFT_MNR_V - Get Time Remaining in Minor Frame

Synopsis

```
retval = VMCALL(HVCALL_TIME_LEFT_MNR_V, p_nsec_left);
```

Description

Get the number of nanoseconds (in Monotonic time) left in the current minor scheduling frame. If the number of nanoseconds exceeds $2^{32}-1$, the call returns $2^{32}-1$.

Arguments

Argument Name	Description
uint64_t p_nsec_left	GVA of a uint32_t variable where the remaining time is stored.

Permissions

None

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_ADDRESS_FAULT	Access to the specified memory location is prohibited by the MMU configuration of the calling Virtual CPU.
LYNXSK_BAD_MEMORY	Access to the specified memory location is prohibited by the security policy set in the configuration vector.
LYNXSK_IO_ERROR	The memory subsystem has reported a hardware error.

TIME_SET - Set Absolute Clock Wall Time

Synopsis

```
retval = VMCALL(HVCALL_TIME_SET, sec, nsec);
```

Description

Sets the current Wall time; the Wall time is the time in seconds and nanoseconds since the Epoch (Jan 1 1970 12:00AM).

Arguments

Argument Name	Description
uint64_t sec	Seconds since the Epoch.
uint64_t nsec	Nanoseconds since the second.

Permissions

```
<absoluteclockflow absclockname=clockname writeabsclockperm="ALLOW"/>
```

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_INVALID_ARG	The time value is invalid: either the nanoseconds are one billion or above, or the seconds are greater than $((2^{63}-1)/10^9-1)$.

VDEV_NOTIFY_PEER - Notify the Virtual Device Peer

Synopsis

```
retval = VMCALL(HVCALL_VDEV_NOTIFY_PEER, vdev_id);
```

Description

A virtual device may be configured with two interfaces, each assigned to a different subject. This hypercall sends a notification interrupt from one of those subjects to the other. The IRQ of the interrupt is as described by the configuration; the receiving subject can find it in the virtual device description structure in its RO page. In fully virtualized subjects, the IRQ determines the input on the virtual interrupt controller activated by the interrupt, rather than maps directly to the injected interrupt vector like in para-virtualized subjects. The notification capability must be enabled for the virtual device interface in the configuration vector.

Arguments

Argument Name	Description
uint32_t vdev_id	The virtual device index in the sending subject's RO page.

Permissions

None

Notes

None

Return Values

Return Value	Description
LYNXSK_SUCCESS	The hypercall has succeeded.
LYNXSK_INVALID_ARG	The virtual device index is invalid, or the virtual device has no peer subject, or the peer interface doesn't have the notification capability.

CHAPTER 6 *Using LynxSecure Features*

This chapter describes how subjects can use LynxSecure® features.

Message Passing Interface

Message passing interface (also known as "message channels" or "message buffers") provides point-to-point, uni-directional, small fixed-size packet (64 bytes), data flows with asynchronous delivery notification between one subject and another. See the section “Message Buffer Flows” in *LynxSecure 6.3.0 Advanced Configuration Guide* for information how to configure a message channel.

Each message channel must have exactly one subject with a write flow to the channel ("sender subject") and exactly one subject with a read flow ("receiver subject"). Sending and retrieving messages is performed via hypercalls, HVCALL_MSG_RECV_V (page 29) and HVCALL_MSG_RECV_V (page 29).

The receiver subject can be configured to receive a synthetic interrupt. Refer to section “Interrupts” (page 6) for information how synthetic interrupts should be handled by a subject.

NOTE: The sender subject does not need to explicitly send an IRQ to the receiver. In fact, sender subject does not even know whether the receiver subject is configured to receive and IRQ or not.

Shared Memory

Memory regions in the HCV may be given flows from more than one subject. In that case, all the subjects which have a flow to that region can access it and the memory region effectively becomes a shared memory region.

It is recommended to use the `SHM` memory type for the memory region. Using other memory types may interfere with LynxSecure use of this region.

It is up to the subjects with the flows to a shared memory region to access it in a coherent manner. The SKH does not provide any protocols or restrictions on what the subjects do with the memory region. The SKH does not dictate what kind of data is populated into that region, nor does it dictate how the data is accessed.

To create a unidirectional stream of data between two subjects, it is possible to use the shared memory region for direct subject-to-subject communication, without any hypervisor involvment. That protocol consists of a ringbuffer with `head` and `tail` indexes, also stored in the shared memory. Ring buffer contains `N` entries (`N` must be power of two). Writer subject checks if there is space in the buffer available (`head - tail < N`), if there is, writes the message into the slot `head % N`, then increments `head`. Reader subject checks if there is data in the buffer (`head != tail`), if there is, reads the message from the slot `tail % N`, then increments `tail`. That is, `head` is only modified by the writer subject, while `tail` is only modified by the reader subject.

Using two such unidirectional streams, it is possible to implement a bidirectional connection between subjects.

Subjects can also use other primitives provided by LynxSecure such as synthetic interrupts to augment the above protocol (e.g. by providing a notification to the reader subject that data is available) or devise a different arbitration protocol.

APPENDIX A Hypercall Permission by System State

Hypercalls are only allowed in specific system states. The following table provides details on the Hypercall permissions associated with the system states.

Table A-1: Hypercalls Permission by State

Hypercalls	System States						
	Validation	Operation	Maintenance Secure	Maintenance Insecure	Initiated BIT	Shutdown	Restart
HVCALL_CHANGE_SCHD_POLICY	A	A	D	D	D	D	D
HVCALL_FLEX_SCHD_GIVE_ALL	A	A	A	A	A	D	D
HVCALL_FLEX_SCHD_GIVE_UNTIL_EVENT	A	A	A	A	A	D	D
HVCALL_FLEX_SCHD_GIVE_UNTIL_SCT	A	A	A	A	A	D	D
HVCALL_FLEX_SCHD_RETURN	A	A	A	A	A	D	D
HVCALL_GET_ROPAGE_INFO	A	A	A	A	A	D	D
HVCALL_GET_SCHD_POLICY_V	A	A	A	A	A	D	D
HVCALL_INTR_EOI	A	A	A	A	A	A	A
HVCALL_MSG_RECV_V	A	A	A	A	A	D	D
HVCALL_MSG_SEND_V	A	A	A	A	A	D	D
HVCALL_RETRIEVE_AUDIT_RECORD_V	A	A	A	A	A	D	D
HVCALL_RETRIEVE_OVERFLOW_AUDIT_RECORD_V	A	A	A	A	A	D	D
HVCALL_ROUTE_INTR	A	A	A	A	A	D	D
HVCALL_SEND_IPI	A	A	A	A	A	D	D
HVCALL_SESM_GET_STATE_V	A	A	A	A	A	D	D
HVCALL_SESM_RESTART_SUBJECT	A	A	A	A	A	D	D
HVCALL_SESM_RESUME_SUBJECT	A	A	A	A	A	D	D
HVCALL_SESM_START_SUBJECT	A	A	A	A	A	D	D
HVCALL_SESM_STOP_SUBJECT	A	A	A	A	A	D	D
HVCALL_SESM_SUSPEND_SUBJECT	A	A	A	A	A	D	D
HVCALL_SET_HRTIMER	A	A	A	A	A	D	D
HVCALL_SET_HRTIMER_VEC	A	A	A	A	A	D	D
HVCALL_SKDB_ENTER	A	A	A	A	A	D	D
HVCALL_SKDB_EXECUTE	A	A	A	A	A	D	D

Hypercalls	System States						
	Validation	Operation	Maintenance Secure	Maintenance Insecure	Initiated BIT	Shutdown	Restart
HVCALL_SSM_GET_STATE_V	A	A	A	A	A	D	D
HVCALL_SSM_SET_INITIATED_BIT	D	A	A	A	D	D	D
HVCALL_SSM_SET_LAST_STATE	D	D	D	D	A	D	D
HVCALL_SSM_SET_MAINTENANCE_INSECURE	A	A	A	D	A	D	D
HVCALL_SSM_SET_MAINTENANCE_SECURE	D	A	D	D	A	D	D
HVCALL_SSM_SET_OPERATION	A	D	A	D	D	D	D
HVCALL_SSM_SET_RESTART	D	A	A	D	D	D	D
HVCALL_SSM_SET_SHUTDOWN	D	A	A	A	D	D	D
HVCALL_STORE_AUDIT_RECORD_V	A	A	A	A	A	D	D
HVCALL_STROBE_WATCHDOG	A	A	A	A	A	D	D
HVCALL_SUBJECT_LOG	A	A	A	A	A	D	D
HVCALL_SYNTH_INTR	A	A	A	A	A	D	D
HVCALL_TCAL_GET_V	A	A	A	A	A	D	D
HVCALL_TCAL_SET	A	A	A	A	A	D	D
HVCALL_TIME_GET_MONOTONIC	A	A	A	A	A	D	D
HVCALL_TIME_GET_V	A	A	A	A	A	D	D
HVCALL_TIME_LEFT_MNR_V	A	A	A	A	A	D	D
HVCALL_TIME_SET	A	A	A	A	A	D	D
HVCALL_VDEV_NOTIFY_PEER	A	A	A	A	A	D	D

Legend: D= Deny; A= Allow

Table A-1 (page 57) specifies the permission status of LynxSecure® hypercalls by state. In general hypercalls are denied for Startup, Shutdown, and Restart and allowed for Operation, Validation and Initiated BIT. Maintenance Insecure is only slightly more restrictive than Operation, denying hypercalls for a return to Operation or a transition to Restart.

APPENDIX B Audit Event Types

The following table lists the Audit Event Types generated by LynxSecure®.

LYNXSK_ADTEVT_CHANGE_SCHD_POLICY_PERM_DENY	The subject did not have the necessary permission to change the scheduling policy using HVCALL_CHANGE_SCHD_POLICY hypercall.
LYNXSK_ADTEVT_CHANGE_SCHD_POLICY_UNKNOWN_POLICY	The subject tried to change the scheduling policy using an invalid policy id for the HVCALL_CHANGE_SCHD_POLICY hypercall.
LYNXSK_ADTEVT_CHANGE_SCHD_POLICY_SUCCESS	The sceduling policy was successfully changed using the HVCALL_CHANGE_SCHD_POLICY hypercall.
LYNXSK_ADTEVT_CHANGE_SCHD_POLICY_ALREADY_IN_PROGRESS	The subject tried to change the scheduling policy using HVCALL_CHANGE_SCHD_POLICY hypercall when a scheduling policy change request was pending.
LYNXSK_ADTEVT_GET_SCHD_PERM_DENY	The subject did not have the necessary permission to get the current scheduling policy using the HVCALL_GET_SCHD_POLICY_V hypercall.
LYNXSK_ADTEVT_GET_ABSOLUTE_TIME_PERM_DENY	The subject did not have the necessary permission to get the absolute time using the HVCALL_TIME_GET_V hypercall.
LYNXSK_ADTEVT_SET_ABSOLUTE_TIME_PERM_DENY	The subject did not have the necessary permission to set the absolute time using the HVCALL_TIME_SET hypercall.
LYNXSK_ADTEVT_SET_ABSOLUTE_TIME_INVALID_TIME	The subject tried to set the absolute time using an invalid value for the nanoseconds using the HVCALL_TIME_SET hypercall.
LYNXSK_ADTEVT_SET_ABSOLUTE_TIME_SUCCESS	The subject was able to successfully set the absolute time using the HVCALL_TIME_SET hypercall.
LYNXSK_ADTEVT_SET_TIME_CALIBRATION_PERM_DENY	The subject did not have the necessary permission to set the timebase calibration value using the HVCALL_TCAL_SET hypercall.
LYNXSK_ADTEVT_SET_TIME_CALIBRATION_INVALID_CALIBRATION	The subject tried to set the timebase calibration value using an invalid argument for the HVCALL_TCAL_SET hypercall.
LYNXSK_ADTEVT_SET_TIME_CALIBRATION_SUCCESS	The subject was able to successfully set the timebase calibration value using the HVCALL_TCAL_SET hypercall.
LYNXSK_ADTEVT_GET_TIME_CALIBRATION_PERM_DENY	The subject did not have the necessary permission to get the timebase calibration value using the HVCALL_TCAL_GET_V hypercall.
LYNXSK_ADTEVT_INIT_TIME	LynxSecure successfully initialized it's internal timekeeping.
LYNXSK_ADTEVT_SESM_STOP_SUBJECT_PERM_DENY	The subject did not have the necessary permission to stop a subject using the HVCALL_SESM_STOP_SUBJECT hypercall.
LYNXSK_ADTEVT_SESM_STOP_SUBJECT_FAILURE	The subject was unable to stop a subject using the HVCALL_SESM_STOP_SUBJECT hypercall because the state transition was invalid.

LYNXSK_ADTEVT_SESM_START_SUBJECT_PERM_DENY	The subject did not have the necessary permission to start a subject using the HVCALL_SESM_START_SUBJECT hypercall.
LYNXSK_ADTEVT_SESM_START_SUBJECT_FAILURE	The subject was unable to start a subject using the HVCALL_SESM_START_SUBJECT hypercall because the state transition was invalid.
LYNXSK_ADTEVT_SESM_SUSPEND_SUBJECT_PERM_DENY	The subject did not have the necessary permission to suspend a subject using the HVCALL_SESM_SUSPEND_SUBJECT hypercall.
LYNXSK_ADTEVT_SESM_SUSPEND_SUBJECT_FAILURE	The subject was unable to suspend a subject using the HVCALL_SESM_SUSPEND_SUBJECT hypercall because the state transition was invalid.
LYNXSK_ADTEVT_SESM_RESUME_SUBJECT_PERM_DENY	The subject did not have the necessary permission to resume a subject using the HVCALL_SESM_RESUME_SUBJECT hypercall.
LYNXSK_ADTEVT_SESM_RESUME_SUBJECT_FAILURE	The subject was unable to resume a subject using the HVCALL_SESM_RESUME_SUBJECT hypercall because the state transition was invalid.
LYNXSK_ADTEVT_SESM_RESTART_SUBJECT_PERM_DENY	The subject did not have the necessary permission to restart a subject using the HVCALL_SESM_RESTART_SUBJECT hypercall.
LYNXSK_ADTEVT_SESM_GET_STATE_PERM_DENY	The subject did not have the necessary permission to get the state of a subject using the HVCALL_SESM_GET_STATE_V hypercall.
LYNXSK_ADTEVT_AUDIT_RETRIEVE_RECORD_PERM_DENY	The subject did not have the necessary permission to retrieve the audit record using the HVCALL_RETRIEVE_AUDIT_RECORD_V hypercall.
LYNXSK_ADTEVT_AUDIT_RETRIEVE_OVERFLOW_PERM_DENY	The subject did not have the necessary permission to retrieve the overflow audit record using the HVCALL_RETRIEVE_OVERFLOW_AUDIT_RECORD_V hypercall.
LYNXSK_ADTEVT_AUDIT_SUBJECT_STORE_RECORD_PERM_DENY	The subject did not have the necessary permission to store an audit record using the HVCALL_STORE_AUDIT_RECORD_V hypercall.
LYNXSK_ADTEVT_AUDIT_SUBJECT_STORE_RECORD_SUCCESS	The subject was able to successfully store an audit record using the HVCALL_STORE_AUDIT_RECORD_V hypercall.
LYNXSK_ADTEVT_MESSAGE_SEND_BAD_MEM	The subject passed an invalid address to the HVCALL_MSG_SEND_V hypercall.
LYNXSK_ADTEVT_MESSAGE_SEND_PERM_DENY	The subject did not have the necessary permission to write to the message channel using the HVCALL_MSG_SEND_V hypercall.
LYNXSK_ADTEVT_MESSAGE_SEND_UNKNOWN_CHANNEL	The subject passed an invalid message channel id as argument to the HVCALL_MSG_SEND_V hypercall.
LYNXSK_ADTEVT_MESSAGE_RECEIVE_BAD_MEM	The subject passed an invalid address to the HVCALL_MSG_RECV_V hypercall.
LYNXSK_ADTEVT_MESSAGE_RECEIVE_PERM_DENY	The subject did not have the necessary permission to read from the message channel using the HVCALL_MSG_RECV_V hypercall.
LYNXSK_ADTEVT_MESSAGE_RECEIVE_UNKNOWN_CHANNEL	The subject passed an invalid message channel id as argument to the HVCALL_MSG_RECV_V hypercall.
LYNXSK_ADTEVT_SEND_SYNTH_INTERRUPT_PERM_DENY	The subject did not have the necessary permission to inject a synthetic interrupt using the HVCALL_SYNTH_INTR hypercall.

LYNXSK_ADTEVT_CHANGE_SCHD_POLICY_FULL_AUDIT_BUFFER	The audit buffer is full and the scheduling policy was changed based on the fullbufferaction entry specified in the HCV.
LYNXSK_ADTEVT_AUDIT_STARTED	LynxSecure successfully initialized the audit subsystem.
LYNXSK_ADTEVT_FULL_AUDIT_BUFFER	The audit buffer is full.
LYNXSK_ADTEVT_SSM_VALIDATION_TRANSITION	The system successfully transitioned to the Validation state.
LYNXSK_ADTEVT_SSM_OPERATION_TRANSITION	The system successfully transitioned to the Operation state.
LYNXSK_ADTEVT_SSM_MAINTENANCE_INSECURE_TRANSITION	The system successfully transitioned to the Maintenance Insecure state.
LYNXSK_ADTEVT_SSM_MAINTENANCE_SECURE_TRANSITION	The system successfully transitioned to the Maintenance Secure state.
LYNXSK_ADTEVT_SSM_IBIT_TRANSITION	The system successfully transitioned to the Initiated BIT state.
LYNXSK_ADTEVT_SSM_GET_STATE_PERM_DENIED	The subject did not have the necessary permission to get the system state using the HVCALL_SSM_GET_STATE_V hypercall.
LYNXSK_ADTEVT_SSM_INITIATED_BIT_PERM_DENIED	The subject did not have the necessary permission to transition the system to Initiated BIT state using the HVCALL_SSM_SET_INITIATED_BIT hypercall.
LYNXSK_ADTEVT_SSM_SET_LAST_STATE_PERM_DENIED	The subject did not have the necessary permission to transition the system to last state entered before the current state using the HVCALL_SSM_SET_LAST_STATE hypercall.
LYNXSK_ADTEVT_SSM_MAINT_INSECURE_PERM_DENIED	The subject did not have the necessary permission to transition the system to Maintenance Insecure state using the HVCALL_SSM_SET_MAINTENANCE_INSECURE hypercall.
LYNXSK_ADTEVT_SSM_MAINT_SECURE_PERM_DENIED	The subject did not have the necessary permission to transition the system to Maintenance Secure state using the HVCALL_SSM_SET_MAINTENANCE_SECURE hypercall.
LYNXSK_ADTEVT_SSM_OPERATION_PERM_DENIED	The subject did not have the necessary permission to transition the system to Operation state using the HVCALL_SSM_SET_OPERATION hypercall.
LYNXSK_ADTEVT_SSM_RESTART_PERM_DENIED	The subject did not have the necessary permission to transition the system to Restart state using the HVCALL_SSM_SET_RESTART hypercall.
LYNXSK_ADTEVT_SSM_SHUTDOWN_PERM_DENIED	The subject did not have the necessary permission to transition the system to Shutdown state using the HVCALL_SSM_SET_SHUTDOWN hypercall.
LYNXSK_ADTEVT_SSM_STARTUP_PERM_DENIED	The subject did not have the necessary permission to transition the system to Startup state using the HVCALL_SSM_SET_STARTUP hypercall.
LYNXSK_ADTEVT_SSM_VALIDATION_PERM_DENIED	The subject did not have the necessary permission to transition the system to Validation state using the HVCALL_SSM_SET_VALIDATION hypercall.
LYNXSK_ADTEVT_SSM_SET_LAST_STATE_FAILURE	The subject failed to transition the system to last state entered before the current state using the HVCALL_SSM_SET_LAST_STATE hypercall.
LYNXSK_ADTEVT_SSM_SET_VALIDATION_FAILURE	The subject failed to transition the system to the Validation state using the HVCALL_SSM_SET_VALIDATION hypercall.

LYNXSK_ADTEVT_SSM_SET_OPERATION_FAILURE	The subject failed to transition the system to the Operation state using the HVCALL_SSM_SET_OPERATION hypercall.
LYNXSK_ADTEVT_SSM_SET_SHUTDOWN_FAILURE	The subject failed to transition the system to the Shutdown state using the HVCALL_SSM_SET_SHUTDOWN hypercall.
LYNXSK_ADTEVT_SSM_SET_STARTUP_FAILURE	The subject failed to transition the system to the Startup state using the HVCALL_SSM_SET_STARTUP hypercall.
LYNXSK_ADTEVT_SSM_SET_RESTART_FAILURE	The subject failed to transition the system to the Restart state using the HVCALL_SSM_SET_RESTART hypercall.
LYNXSK_ADTEVT_SSM_SET_IBIT_FAILURE	The subject failed to transition the system to the Initiated BIT state using the HVCALL_SSM_SET_INITIATED_BIT hypercall.
LYNXSK_ADTEVT_SSM_SET_MAINT_SECURE_FAILURE	The subject failed to transition the system to the Maintenance Secure state using the HVCALL_SSM_SET_MAINTENANCE_SECURE hypercall.
LYNXSK_ADTEVT_SSM_SET_MAINT_INSECURE_FAILURE	The subject failed to transition the system to the Maintenance Insecure state using the HVCALL_SSM_SET_MAINTENANCE_INSECURE hypercall.
LYNXSK_ADTEVT_FLEX_SCHD_RETURN_PERM_DENY	The subject did not have the necessary permission to return CPU time to the nominally scheduled subject using the HVCALL_FLEX_SCHD_RETURN hypercall.
LYNXSK_ADTEVT_FLEX_SCHD_GIVE_ALL_PERM_DENY	The subject did not have the necessary permission to donate time to the specified subject indefinitely using the HVCALL_FLEX_SCHD_GIVE_ALL hypercall.
LYNXSK_ADTEVT_FLEX_SCHD_GIVE_SCT_PERM_DENY	The subject did not have the necessary permission to donate current subject's CPU time until the next system clock tick to the specified subject using the HVCALL_FLEX_SCHD_GIVE_UNTIL_SCT hypercall.
LYNXSK_ADTEVT_FLEX_SCHD_GIVE_UNTIL_EVENT_PERM_DENY	The subject did not have the necessary permission to donate current subject's CPU time until the next asynchronous event to the specified subject using the HVCALL_FLEX_SCHD_GIVE_UNTIL_EVENT hypercall.
LYNXSK_ADTEVT_MM_POLICY_VIOLATION	The subject violated memory manager policy.
LYNXSK_ADTEVT_MM_BAD_MMU_MODE	A Para Virtualized subject tried to change their page table mode or WP flag.
LYNXSK_ADTEVT_IOMMU_POLICY_VIOLATION	The LynxSecure IOMMU subsystem encountered a fault due to a read or write being denied.
LYNXSK_ADTEVT_UNKNOWN_HVCALL_PERM_DENIED	An unsupported hypercall was made from the subject.
LYNXSK_ADTEVT_HRTMR_SET_VEC_INVALID_VEC	An invalid interrupt vector was passed as argument by the subject using the HVCALL_SET_HRTIMER_VEC hypercall.
LYNXSK_ADTEVT_ROUTE_INTR_INVALID_ARG	An invalid interrupt vector was passed as argument by the subject using the HVCALL_ROUTE_INTR hypercall.
LYNXSK_ADTEVT_ROUTE_INTR_ADDR_FAULT	There was a problem with virtual address translation by LynxSecure for an argument passed by the subject using the HVCALL_ROUTE_INTR hypercall.
LYNXSK_ADTEVT_ROUTE_INTR_BAD_MEM	There was a problem with memory management security policy violation or a failure that might have led to a loss of secure state for an argument passed by the subject using the HVCALL_ROUTE_INTR hypercall.
LYNXSK_ADTEVT_ROUTE_INTR_FAILURE	There was a problem with routing interrupts by the subject using the HVCALL_ROUTE_INTR hypercall because of an invalid vector.

LYNXSK_ADTEVT_SEND_IPI_INVALID_SUBJECT	An invalid virtual CPU identifier was passed as argument by the subject using the HVCALL_SEND_IPI hypercall.
LYNXSK_ADTEVT_SEND_IPI_INVALID_VEC	An invalid interrupt vector was passed as argument by the subject using the HVCALL_SEND_IPI hypercall.
LYNXSK_ADTEVT_VRM_REGISTER_EP_BAD_MEM	There was a memory management security policy violation or a failure that might have led to a loss of secure state for LynxSecure by an argument passed by the subject using the HVCALL_REGISTER_SUBJECT_VRM hypercall.
LYNXSK_ADTEVT_LEAVE_VRM_INVALID_CTX	There was an invalid hypercall made by the subject, HVCALL_LEAVE_VRM without a corresponding HVCALL_REGISTER_SUBJECT_VRM hypercall.

APPENDIX C API Changes

This appendix lists API changes over LynxSecure® release history that are not backward-compatible with previous releases. The user needs to be aware of these changes when porting software between LynxSecure releases.

Release	Change Description
Post-5.2	The Read-Only Page structure `rmr_t` field `memory_type` for memory regions configured with the types `SHM`, `SANSRC` and `SANDST` in the HCV used to have the `MEM_TYPE_PROGRAM` bit set. For example, the value of the `memory_type` field for a `SHM` memory region was `MEM_TYPE_SHM | MEM_TYPE_PROGRAM`. This is no longer the case. For example, the value of the `memory_type` field for a `SHM` memory region is `MEM_TYPE_SHM`.
6.0.0	The RO page `mem_maps` array, whose meaning was different in PV and FV subjects, has been split into the `phys_maps` and `virt_maps` arrays with a fixed meaning regardless of the subject type.
6.0.0	Hypercall-based High-Resolution timers bound to the Wall time have been removed. The only type of hypercall-based HR timers available is relative to the current Monotonic time.
6.0.0	Support for x86 systems where the CPU doesn't support the Extended Page Table has been discontinued. Because of this, PV subject page table manipulation hypercalls are no longer necessary and have been removed.