Protected Execution Facility

Contents

Introduction

Protected Execution Facility (PEF) is an architectural change for POWER 9 that enables Secure Virtual Machines (SVMs). DD2.3 chips (PVR=0x004e1203) or greater will be PEF-capable. A new ISA release will include the PEF RFC02487 changes.

When enabled, PEF adds a new higher privileged mode, called Ultravisor mode, to POWER architecture. Along with the new mode there is new firmware called the Protected Execution Ultravisor (or Ultravisor for short). Ultravisor mode is the highest privileged mode in POWER architecture.

Privilege States
Problem
Supervisor
Hypervisor
Ultravisor

PEF protects SVMs from the hypervisor, privileged users, and other VMs in the system. SVMs are protected while at rest and can only be executed by an authorized machine. All virtual machines utilize hypervisor services. The Ultravisor filters calls between the SVMs and the hypervisor to assure that information does not accidentally leak. All hypercalls except H_RANDOM are reflected to the hypervisor. H_RANDOM is not reflected to prevent the hypervisor from influencing random values in the SVM.

To support this there is a refactoring of the ownership of resources in the CPU. Some of the resources which were previously hypervisor privileged are now ultravisor privileged.

Hardware

The hardware changes include the following:

- There is a new bit in the MSR that determines whether the current process is running in secure mode, MSR(S) bit 41. MSR(S)=1, process is in secure mode, MSR(s)=0 process is in normal mode.
- The MSR(S) bit can only be set by the Ultravisor.
- HRFID cannot be used to set the MSR(S) bit. If the hypervisor needs to return to a SVM it must use an ultracall. It

can determine if the VM it is returning to is secure.

- There is a new Ultravisor privileged register, SMFCTRL, which has an enable/disable bit SMFCTRL(E).
- The privilege of a process is now determined by three MSR bits, MSR(S, HV, PR). In each of the tables below the modes are listed from least privilege to highest privilege. The higher privilege modes can access all the resources of the lower privilege modes.

  Secure Mode MSR Settings

S	HV	PR	Privilege
1	0	1	Problem
1	0	0	Privileged(OS)
1	1	0	Ultravisor
1	1	1	Reserved

  Normal Mode MSR Settings

S	HV	PR	Privilege
0	0	1	Problem
0	0	0	Privileged(OS)
0	1	0	Hypervisor
0	1	1	Problem (Host)

- Memory is partitioned into secure and normal memory. Only processes that are running in secure mode can access secure memory.
- The hardware does not allow anything that is not running secure to access secure memory. This means that the Hypervisor cannot access the memory of the SVM without using an ultracall (asking the Ultravisor). The Ultravisor will only allow the hypervisor to see the SVM memory encrypted.
- I/O systems are not allowed to directly address secure memory. This limits the SVMs to virtual I/O only.
- The architecture allows the SVM to share pages of memory with the hypervisor that are not protected with encryption. However, this sharing must be initiated by the SVM.
- When a process is running in secure mode all hypercalls (syscall lev=1) go to the Ultravisor.
- When a process is in secure mode all interrupts go to the Ultravisor.
- The following resources have become Ultravisor privileged and require an Ultravisor interface to manipulate:
  - Processor configurations registers (SCOMs).
  - Stop state information.
  - The debug registers CIABR, DAWR, and DAWRX when SMFCTRL(D) is set. If SMFCTRL(D) is not set they do not work in secure mode. When set, reading and writing requires an Ultravisor call, otherwise that will cause a Hypervisor Emulation Assistance interrupt.
  - PTCR and partition table entries (partition table is in secure memory). An attempt to write to PTCR will cause a Hypervisor Emulation Assitance interrupt.
  - LDBAR (LD Base Address Register) and IMC (In-Memory Collection) non-architected registers. An attempt to write to them will cause a Hypervisor Emulation Assistance interrupt.
  - Paging for an SVM, sharing of memory with Hypervisor for an SVM. (Including Virtual Processor Area (VPA) and virtual I/O).

Software/Microcode

The software changes include:

- SVMs are created from normal VM using (open source) tooling supplied by IBM.
- All SVMs start as normal VMs and utilize an ultracall, UV_ESM (Enter Secure Mode), to make the transition.
- When the UV_ESM ultracall is made the Ultravisor copies the VM into secure memory, decrypts the verification information, and checks the integrity of the SVM. If the integrity check passes the Ultravisor passes control in secure mode.
- The verification information includes the pass phrase for the encrypted disk associated with the SVM. This pass phrase is given to the SVM when requested.
- The Ultravisor is not involved in protecting the encrypted disk of the SVM while at rest.
- For external interrupts the Ultravisor saves the state of the SVM, and reflects the interrupt to the hypervisor for processing. For hypercalls, the Ultravisor inserts neutral state into all registers not needed for the hypercall then reflects the call to the hypervisor for processing. The H_RANDOM hypercall is performed by the Ultravisor and not reflected.
- For virtual I/O to work bounce buffering must be done.
- The Ultravisor uses AES (IAPM) for protection of SVM memory. IAPM is a mode of AES that provides integrity and secrecy concurrently.

- The movement of data between normal and secure pages is coordinated with the Ultravisor by a new HMM plug-in in the Hypervisor.

The Ultravisor offers new services to the hypervisor and SVMs. These are accessed through ultracalls.

Terminology

- Hypercalls: special system calls used to request services from Hypervisor.
- Normal memory: Memory that is accessible to Hypervisor.
- Normal page: Page backed by normal memory and available to Hypervisor.
- Shared page: A page backed by normal memory and available to both the Hypervisor/QEMU and the SVM (i.e page has mappings in SVM and Hypervisor/QEMU).
- Secure memory: Memory that is accessible only to Ultravisor and SVMs.
- Secure page: Page backed by secure memory and only available to Ultravisor and SVM.
- SVM: Secure Virtual Machine.
- Ultracalls: special system calls used to request services from Ultravisor.

Ultravisor calls API

This section describes Ultravisor calls (ultracalls) needed to support Secure Virtual Machines (SVM)s and Paravirtualized KVM. The ultracalls allow the SVMs and Hypervisor to request services from the Ultravisor such as accessing a register or memory region that can only be accessed when running in Ultravisor-privileged mode.

The specific service needed from an ultracall is specified in register R3 (the first parameter to the ultracall). Other parameters to the ultracall, if any, are specified in registers R4 through R12.

Return value of all ultracalls is in register R3. Other output values from the ultracall, if any, are returned in registers R4 through R12. The only exception to this register usage is the `UV_RETURN` ultracall described below.

Each ultracall returns specific error codes, applicable in the context of the ultracall. However, like with the PowerPC Architecture Platform Reference (PAPR), if no specific error code is defined for a particular situation, then the ultracall will fallback to an erroneous parameter-position based code. i.e U_PARAMETER, U_P2, U_P3 etc depending on the ultracall parameter that may have caused the error.

Some ultracalls involve transferring a page of data between Ultravisor and Hypervisor. Secure pages that are transferred from secure memory to normal memory may be encrypted using dynamically generated keys. When the secure pages are transferred back to secure memory, they may be decrypted using the same dynamically generated keys. Generation and management of these keys will be covered in a separate document.

For now this only covers ultracalls currently implemented and being used by Hypervisor and SVMs but others can be added here when it makes sense.

The full specification for all hypercalls/ultracalls will eventually be made available in the public/OpenPower version of the PAPR specification.

> **Note**
>
> If PEF is not enabled, the ultracalls will be redirected to the Hypervisor which must handle/fail the calls.

Ultracalls used by Hypervisor

This section describes the virtual memory management ultracalls used by the Hypervisor to manage SVMs.

UV_PAGE_OUT

Encrypt and move the contents of a page from secure memory to normal memory.

Syntax

```
uint64_t ultracall(const uint64_t UV_PAGE_OUT,
        uint16_t lpid,          /* LPAR ID */
        uint64_t dest_ra,       /* real address of destination page */
        uint64_t src_gpa,       /* source guest-physical-address */
        uint8_t  flags,         /* flags */
        uint64_t order)         /* page size order */
```

Return values

One of the following values:

- U_SUCCESS on success.

- U_PARAMETER if `lpid` is invalid.
- U_P2 if `dest_ra` is invalid.
- U_P3 if the `src_gpa` address is invalid.
- U_P4 if any bit in the `flags` is unrecognized
- U_P5 if the `order` parameter is unsupported.
- U_FUNCTION if functionality is not supported.
- U_BUSY if page cannot be currently paged-out.

Description

Encrypt the contents of a secure-page and make it available to Hypervisor in a normal page.

By default, the source page is unmapped from the SVM's partition- scoped page table. But the Hypervisor can provide a hint to the Ultravisor to retain the page mapping by setting the `UV_SNAPSHOT` flag in `flags` parameter.

If the source page is already a shared page the call returns U_SUCCESS, without doing anything.

Use cases

1. QEMU attempts to access an address belonging to the SVM but the page frame for that address is not mapped into QEMU's address space. In this case, the Hypervisor will allocate a page frame, map it into QEMU's address space and issue the `UV_PAGE_OUT` call to retrieve the encrypted contents of the page.
2. When Ultravisor runs low on secure memory and it needs to page-out an LRU page. In this case, Ultravisor will issue the `H_SVM_PAGE_OUT` hypercall to the Hypervisor. The Hypervisor will then allocate a normal page and issue the `UV_PAGE_OUT` ultracall and the Ultravisor will encrypt and move the contents of the secure page into the normal page.
3. When Hypervisor accesses SVM data, the Hypervisor requests the Ultravisor to transfer the corresponding page into a insecure page, which the Hypervisor can access. The data in the normal page will be encrypted though.

UV_PAGE_IN

Move the contents of a page from normal memory to secure memory.

Syntax

```
uint64_t ultracall(const uint64_t UV_PAGE_IN,
        uint16_t lpid,          /* the LPAR ID */
        uint64_t src_ra,        /* source real address of page */
        uint64_t dest_gpa,      /* destination guest physical address */
        uint64_t flags,         /* flags */
        uint64_t order)         /* page size order */
```

Return values

One of the following values:

- U_SUCCESS on success.
- U_BUSY if page cannot be currently paged-in.
- U_FUNCTION if functionality is not supported
- U_PARAMETER if `lpid` is invalid.
- U_P2 if `src_ra` is invalid.
- U_P3 if the `dest_gpa` address is invalid.
- U_P4 if any bit in the `flags` is unrecognized
- U_P5 if the `order` parameter is unsupported.

Description

Move the contents of the page identified by `src_ra` from normal memory to secure memory and map it to the guest physical address `dest_gpa`.

If *dest_gpa* refers to a shared address, map the page into the partition-scoped page-table of the SVM. If *dest_gpa* is not shared, copy the contents of the page into the corresponding secure page. Depending on the context, decrypt the page before being copied.

The caller provides the attributes of the page through the `flags` parameter. Valid values for `flags` are:

- CACHE_INHIBITED
- CACHE_ENABLED
- WRITE_PROTECTION

The Hypervisor must pin the page in memory before making `UV_PAGE_IN` ultracall.

Use cases

1. When a normal VM switches to secure mode, all its pages residing in normal memory, are moved into secure memory.
2. When an SVM requests to share a page with Hypervisor the Hypervisor allocates a page and informs the Ultravisor.
3. When an SVM accesses a secure page that has been paged-out, Ultravisor invokes the Hypervisor to locate the page. After locating the page, the Hypervisor uses UV_PAGE_IN to make the page available to Ultravisor.

UV_PAGE_INVAL

Invalidate the Ultravisor mapping of a page.

Syntax

```
uint64_t ultracall(const uint64_t UV_PAGE_INVAL,
        uint16_t lpid,          /* the LPAR ID */
        uint64_t guest_pa,      /* destination guest-physical-address */
        uint64_t order)         /* page size order */
```

Return values

One of the following values:

- U_SUCCESS on success.
- U_PARAMETER if `lpid` is invalid.
- U_P2 if `guest_pa` is invalid (or corresponds to a secure page mapping).
- U_P3 if the `order` is invalid.
- U_FUNCTION if functionality is not supported.
- U_BUSY if page cannot be currently invalidated.

Description

This ultracall informs Ultravisor that the page mapping in Hypervisor corresponding to the given guest physical address has been invalidated and that the Ultravisor should not access the page. If the specified `guest_pa` corresponds to a secure page, Ultravisor will ignore the attempt to invalidate the page and return U_P2.

Use cases

1. When a shared page is unmapped from the QEMU's page table, possibly because it is paged-out to disk, Ultravisor needs to know that the page should not be accessed from its side too.

UV_WRITE_PATE

Validate and write the partition table entry (PATE) for a given partition.

Syntax

```
uint64_t ultracall(const uint64_t UV_WRITE_PATE,
        uint32_t lpid,          /* the LPAR ID */
        uint64_t dw0            /* the first double word to write */
        uint64_t dw1)           /* the second double word to write */
```

Return values

One of the following values:

- U_SUCCESS on success.
- U_BUSY if PATE cannot be currently written to.
- U_FUNCTION if functionality is not supported.
- U_PARAMETER if `lpid` is invalid.
- U_P2 if `dw0` is invalid.
- U_P3 if the `dw1` address is invalid.
- U_PERMISSION if the Hypervisor is attempting to change the PATE of a secure virtual machine or if called from a context other than Hypervisor.

Description

Validate and write a LPID and its partition-table-entry for the given LPID. If the LPID is already allocated and initialized, this call results in changing the partition table entry.

Use cases

1. The Partition table resides in Secure memory and its entries, called PATE (Partition Table Entries), point to the partition- scoped page tables for the Hypervisor as well as each of the virtual machines (both secure and normal). The Hypervisor operates in partition 0 and its partition-scoped page tables reside in normal memory.
2. This ultracall allows the Hypervisor to register the partition- scoped and process-scoped page table entries for the Hypervisor and other partitions (virtual machines) with the Ultravisor.
3. If the value of the PATE for an existing partition (VM) changes, the TLB cache for the partition is flushed.
4. The Hypervisor is responsible for allocating LPID. The LPID and its PATE entry are registered together. The Hypervisor manages the PATE entries for a normal VM and can change the PATE entry anytime. Ultravisor manages the PATE entries for an SVM and Hypervisor is not allowed to modify them.

UV_RETURN

Return control from the Hypervisor back to the Ultravisor after processing an hypercall or interrupt that was forwarded (aka *reflected*) to the Hypervisor.

Syntax

```
uint64_t ultracall(const uint64_t UV_RETURN)
```

Return values

This call never returns to Hypervisor on success. It returns U_INVALID if ultracall is not made from a Hypervisor context.

Description

When an SVM makes an hypercall or incurs some other exception, the Ultravisor usually forwards (aka *reflects*) the exceptions to the Hypervisor. After processing the exception, Hypervisor uses the `UV_RETURN` ultracall to return control back to the SVM.

The expected register state on entry to this ultracall is:

- Non-volatile registers are restored to their original values.
- If returning from an hypercall, register R0 contains the return value (**unlike other ultracalls**) and, registers R4 through R12 contain any output values of the hypercall.
- R3 contains the ultracall number, i.e UV_RETURN.
- If returning with a synthesized interrupt, R2 contains the synthesized interrupt number.

Use cases

1. Ultravisor relies on the Hypervisor to provide several services to the SVM such as processing hypercall and other exceptions. After processing the exception, Hypervisor uses UV_RETURN to return control back to the Ultravisor.
2. Hypervisor has to use this ultracall to return control to the SVM.

UV_REGISTER_MEM_SLOT

Register an SVM address-range with specified properties.

Syntax

```
uint64_t ultracall(const uint64_t UV_REGISTER_MEM_SLOT,
        uint64_t lpid,          /* LPAR ID of the SVM */
        uint64_t start_gpa,     /* start guest physical address */
        uint64_t size,          /* size of address range in bytes */
        uint64_t flags          /* reserved for future expansion */
        uint16_t slotid)        /* slot identifier */
```

Return values

One of the following values:

- U_SUCCESS on success.
- U_PARAMETER if `lpid` is invalid.
- U_P2 if `start_gpa` is invalid.
- U_P3 if `size` is invalid.

- U_P4 if any bit in the `flags` is unrecognized.
- U_P5 if the `slotid` parameter is unsupported.
- U_PERMISSION if called from context other than Hypervisor.
- U_FUNCTION if functionality is not supported.

Description

Register a memory range for an SVM. The memory range starts at the guest physical address `start_gpa` and is `size` bytes long.

Use cases

1. When a virtual machine goes secure, all the memory slots managed by the Hypervisor move into secure memory. The Hypervisor iterates through each of memory slots, and registers the slot with Ultravisor. Hypervisor may discard some slots such as those used for firmware (SLOF).
2. When new memory is hot-plugged, a new memory slot gets registered.

UV_UNREGISTER_MEM_SLOT

Unregister an SVM address-range that was previously registered using UV_REGISTER_MEM_SLOT.

Syntax

```
uint64_t ultracall(const uint64_t UV_UNREGISTER_MEM_SLOT,
        uint64_t lpid,          /* LPAR ID of the SVM */
        uint64_t slotid)        /* reservation slotid */
```

Return values

One of the following values:

- U_SUCCESS on success.
- U_FUNCTION if functionality is not supported.
- U_PARAMETER if `lpid` is invalid.
- U_P2 if `slotid` is invalid.
- U_PERMISSION if called from context other than Hypervisor.

Description

Release the memory slot identified by `slotid` and free any resources allocated towards the reservation.

Use cases

1. Memory hot-remove.

UV_SVM_TERMINATE

Terminate an SVM and release its resources.

Syntax

```
uint64_t ultracall(const uint64_t UV_SVM_TERMINATE,
        uint64_t lpid,          /* LPAR ID of the SVM */)
```

Return values

One of the following values:

- U_SUCCESS on success.
- U_FUNCTION if functionality is not supported.
- U_PARAMETER if `lpid` is invalid.
- U_INVALID if VM is not secure.
- U_PERMISSION if not called from a Hypervisor context.

Description

Terminate an SVM and release all its resources.

Use cases

1. Called by Hypervisor when terminating an SVM.

Ultracalls used by SVM

UV_SHARE_PAGE

Share a set of guest physical pages with the Hypervisor.

Syntax

```
uint64_t ultracall(const uint64_t UV_SHARE_PAGE,
        uint64_t gfn,   /* guest page frame number */
        uint64_t num)   /* number of pages of size PAGE_SIZE */
```

Return values

One of the following values:

- U_SUCCESS on success.
- U_FUNCTION if functionality is not supported.
- U_INVALID if the VM is not secure.
- U_PARAMETER if `gfn` is invalid.
- U_P2 if `num` is invalid.

Description

Share the `num` pages starting at guest physical frame number `gfn` with the Hypervisor. Assume page size is PAGE_SIZE bytes. Zero the pages before returning.

If the address is already backed by a secure page, unmap the page and back it with an insecure page, with the help of the Hypervisor. If it is not backed by any page yet, mark the PTE as insecure and back it with an insecure page when the address is accessed. If it is already backed by an insecure page, zero the page and return.

Use cases

1. The Hypervisor cannot access the SVM pages since they are backed by secure pages. Hence an SVM must explicitly request Ultravisor for pages it can share with Hypervisor.
2. Shared pages are needed to support virtio and Virtual Processor Area (VPA) in SVMs.

UV_UNSHARE_PAGE

Restore a shared SVM page to its initial state.

Syntax

```
uint64_t ultracall(const uint64_t UV_UNSHARE_PAGE,
        uint64_t gfn,   /* guest page frame number */
        uint73 num)     /* number of pages of size PAGE_SIZE*/
```

Return values

One of the following values:

- U_SUCCESS on success.
- U_FUNCTION if functionality is not supported.
- U_INVALID if VM is not secure.
- U_PARAMETER if `gfn` is invalid.
- U_P2 if `num` is invalid.

Description

Stop sharing `num` pages starting at `gfn` with the Hypervisor. Assume that the page size is PAGE_SIZE. Zero the pages before returning.

If the address is already backed by an insecure page, unmap the page and back it with a secure page. Inform the Hypervisor to release reference to its shared page. If the address is not backed by a page yet, mark the PTE as secure and back it with a secure page when that address is accessed. If it is already backed by an secure page zero the page and return.

Use cases

1. The SVM may decide to unshare a page from the Hypervisor.

UV_UNSHARE_ALL_PAGES

Unshare all pages the SVM has shared with Hypervisor.

Syntax

```
uint64_t ultracall(const uint64_t UV_UNSHARE_ALL_PAGES)
```

Return values

One of the following values:

- U_SUCCESS on success.
- U_FUNCTION if functionality is not supported.
- U_INVAL if VM is not secure.

Description

Unshare all shared pages from the Hypervisor. All unshared pages are zeroed on return. Only pages explicitly shared by the SVM with the Hypervisor (using UV_SHARE_PAGE ultracall) are unshared. Ultravisor may internally share some pages with the Hypervisor without explicit request from the SVM. These pages will not be unshared by this ultracall.

Use cases

1. This call is needed when `kexec` is used to boot a different kernel. It may also be needed during SVM reset.

UV_ESM

Secure the virtual machine (*enter secure mode*).

Syntax

```
uint64_t ultracall(const uint64_t UV_ESM,
        uint64_t esm_blob_addr, /* location of the ESM blob */
        unint64_t fdt)          /* Flattened device tree */
```

Return values

One of the following values:

- U_SUCCESS on success (including if VM is already secure).
- U_FUNCTION if functionality is not supported.
- U_INVALID if VM is not secure.
- U_PARAMETER if `esm_blob_addr` is invalid.
- U_P2 if `fdt` is invalid.
- U_PERMISSION if any integrity checks fail.
- U_RETRY insufficient memory to create SVM.
- U_NO_KEY symmetric key unavailable.

Description

Secure the virtual machine. On successful completion, return control to the virtual machine at the address specified in the ESM blob.

Use cases

1. A normal virtual machine can choose to switch to a secure mode.

Hypervisor Calls API

This document describes the Hypervisor calls (hypercalls) that are needed to support the Ultravisor. Hypercalls are services provided by the Hypervisor to virtual machines and Ultravisor.

Register usage for these hypercalls is identical to that of the other hypercalls defined in the Power Architecture Platform Reference (PAPR) document. i.e on input, register R3 identifies the specific service that is being requested and registers R4 through R11 contain additional parameters to the hypercall, if any. On output, register R3 contains the return value and registers R4 through R9 contain any other output values from the hypercall.

This document only covers hypercalls currently implemented/planned for Ultravisor usage but others can be added here when it makes sense.

The full specification for all hypercalls/ultracalls will eventually be made available in the public/OpenPower version of the PAPR specification.

Hypervisor calls to support Ultravisor

Following are the set of hypercalls needed to support Ultravisor.

H_SVM_INIT_START

Begin the process of converting a normal virtual machine into an SVM.

Syntax

```
uint64_t hypercall(const uint64_t H_SVM_INIT_START)
```

Return values

One of the following values:

- H_SUCCESS on success.
- H_STATE if the VM is not in a position to switch to secure.

Description

Initiate the process of securing a virtual machine. This involves coordinating with the Ultravisor, using ultracalls, to allocate resources in the Ultravisor for the new SVM, transferring the VM's pages from normal to secure memory etc. When the process is completed, Ultravisor issues the H_SVM_INIT_DONE hypercall.

Use cases

1. Ultravisor uses this hypercall to inform Hypervisor that a VM has initiated the process of switching to secure mode.

H_SVM_INIT_DONE

Complete the process of securing an SVM.

Syntax

```
uint64_t hypercall(const uint64_t H_SVM_INIT_DONE)
```

Return values

One of the following values:

- H_SUCCESS on success.
- H_UNSUPPORTED if called from the wrong context (e.g. from an SVM or before an H_SVM_INIT_START hypercall).
- H_STATE if the hypervisor could not successfully transition the VM to Secure VM.

Description

Complete the process of securing a virtual machine. This call must be made after a prior call to `H_SVM_INIT_START` hypercall.

Use cases

On successfully securing a virtual machine, the Ultravisor informs Hypervisor about it. Hypervisor can use this call to finish setting up its internal state for this virtual machine.

H_SVM_INIT_ABORT

Abort the process of securing an SVM.

Syntax

```
uint64_t hypercall(const uint64_t H_SVM_INIT_ABORT)
```

Return values

One of the following values:

- H_PARAMETER on successfully cleaning up the state,
  Hypervisor will return this value to the **guest**, to indicate that the underlying UV_ESM ultracall failed.
- H_STATE if called after a VM has gone secure (i.e
  H_SVM_INIT_DONE hypercall was successful).
- H_UNSUPPORTED if called from a wrong context (e.g. from a
  normal VM).

Description

Abort the process of securing a virtual machine. This call must be made after a prior call to `H_SVM_INIT_START` hypercall and before a call to `H_SVM_INIT_DONE`.

On entry into this hypercall the non-volatile GPRs and FPRs are expected to contain the values they had at the time the VM issued the UV_ESM ultracall. Further `SRR0` is expected to contain the address of the instruction after the `UV_ESM` ultracall and `SRR1` the MSR value with which to return to the VM.

This hypercall will cleanup any partial state that was established for the VM since the prior `H_SVM_INIT_START` hypercall, including paging out pages that were paged-into secure memory, and issue the `UV_SVM_TERMINATE` ultracall to terminate the VM.

After the partial state is cleaned up, control returns to the VM (**not Ultravisor**), at the address specified in `SRR0` with the MSR values set to the value in `SRR1`.

Use cases

If after a successful call to `H_SVM_INIT_START`, the Ultravisor encounters an error while securing a virtual machine, either due to lack of resources or because the VM's security information could not be validated, Ultravisor informs the Hypervisor about it. Hypervisor should use this call to clean up any internal state for this virtual machine and return to the VM.

H_SVM_PAGE_IN

Move the contents of a page from normal memory to secure memory.

Syntax

```
uint64_t hypercall(const uint64_t H_SVM_PAGE_IN,
        uint64_t guest_pa,      /* guest-physical-address */
        uint64_t flags,         /* flags */
        uint64_t order)         /* page size order */
```

Return values

One of the following values:

- H_SUCCESS on success.
- H_PARAMETER if `guest_pa` is invalid.
- H_P2 if `flags` is invalid.
- H_P3 if `order` of page is invalid.

Description

Retrieve the content of the page, belonging to the VM at the specified guest physical address.

Only valid value(s) in `flags` are:

- H_PAGE_IN_SHARED which indicates that the page is to be shared with the Ultravisor.
- H_PAGE_IN_NONSHARED indicates that the UV is not anymore interested in the page. Applicable if the page is a shared page.

The `order` parameter must correspond to the configured page size.

Use cases

1. When a normal VM becomes a secure VM (using the UV_ESM ultracall), the Ultravisor uses this hypercall to move contents of each page of the VM from normal memory to secure memory.
2. Ultravisor uses this hypercall to ask Hypervisor to provide a page in normal memory that can be shared between the SVM and Hypervisor.
3. Ultravisor uses this hypercall to page-in a paged-out page. This can happen when the SVM touches a paged-out page.
4. If SVM wants to disable sharing of pages with Hypervisor, it can inform Ultravisor to do so. Ultravisor will then use this hypercall and inform Hypervisor that it has released access to the normal page.

H_SVM_PAGE_OUT

Move the contents of the page to normal memory.

Syntax

```
uint64_t hypercall(const uint64_t H_SVM_PAGE_OUT,
        uint64_t guest_pa,      /* guest-physical-address */
        uint64_t flags,         /* flags (currently none) */
        uint64_t order)         /* page size order */
```

Return values

One of the following values:

- H_SUCCESS on success.
- H_PARAMETER if `guest_pa` is invalid.
- H_P2 if `flags` is invalid.
- H_P3 if `order` is invalid.

Description

Move the contents of the page identified by `guest_pa` to normal memory.

Currently `flags` is unused and must be set to 0. The `order` parameter must correspond to the configured page size.

Use cases

1. If Ultravisor is running low on secure pages, it can move the contents of some secure pages, into normal pages using this hypercall. The content will be encrypted.

References

- Supporting Protected Computing on IBM Power Architecture