Hypercall Op-codes (hcalls)

Overview

Virtualization on 64-bit Power Book3S Platforms is based on the PAPR specification [1] which describes the run-time environment for a guest operating system and how it should interact with the hypervisor for privileged operations. Currently there are two PAPR compliant hypervisors:

- **IBM PowerVM (PHYP)**: IBM's proprietary hypervisor that supports AIX, IBM-i and Linux as supported guests (termed as Logical Partitions or LPARS). It supports the full PAPR specification.
- **Qemu/KVM**: Supports PPC64 linux guests running on a PPC64 linux host. Though it only implements a subset of PAPR specification called LoPAPR [2].

On PPC64 arch a guest kernel running on top of a PAPR hypervisor is called a *pSeries guest*. A pseries guest runs in a supervisor mode (HV=0) and must issue hypercalls to the hypervisor whenever it needs to perform an action that is hypervisor priviledged [3] or for other services managed by the hypervisor.

Hence a Hypercall (hcall) is essentially a request by the pseries guest asking hypervisor to perform a privileged operation on behalf of the guest. The guest issues a with necessary input operands. The hypervisor after performing the privilege operation returns a status code and output operands back to the guest.

HCALL ABI

The ABI specification for a hcall between a pseries guest and PAPR hypervisor is covered in section 14.5.3 of ref [2]. Switch to the Hypervisor context is done via the instruction **HVCS** that expects the Opcode for hcall is set in *r3* and any in-arguments for the hcall are provided in registers *r4-r12*. If values have to be passed through a memory buffer, the data stored in that buffer should be in Big-endian byte order.

Once control returns back to the guest after hypervisor has serviced the 'HVCS' instruction the return value of the hcall is available in *r3* and any out values are returned in registers *r4-r12*. Again like in case of in-arguments, any out values stored in a memory buffer will be in Big-endian byte order.

Powerpc arch code provides convenient wrappers named **plpar_hcall_xxx** defined in a arch specific header [4] to issue hcalls from the linux kernel running as pseries guest.

Register Conventions

Any hcall should follow same register convention as described in section 2.2.1.1 of "64-Bit ELF V2 ABI Specification: Power Architecture"[5]. Table below summarizes these conventions:

Register Range	Volatile (Y/N)	Purpose
r0	Y	Optional-usage
r1	N	Stack Pointer
r2	N	TOC
r3	Y	hcall opcode/return value
r4-r10	Y	in and out values
r11	Y	Optional-usage/Environmental pointer
r12	Y	Optional-usage/Function entry address at global entry point
r13	N	Thread-Pointer
r14-r31	N	Local Variables
LR	Y	Link Register
CTR	Y	Loop Counter
XER	Y	Fixed-point exception register.
CR0-1	Y	Condition register fields.
CR2-4	N	Condition register fields.
CR5-7	Y	Condition register fields.
Others	N	

DRC & DRC Indexes

```
 DR1                                    Guest
 +--+        +------------+         +---------+
 |  | <----> |            |         |  User   |
 +--+  DRC1  |            |   DRC   |  Space  |
             |    PAPR    |  Index  +---------+
 DR2         | Hypervisor |         |         |
```

```
+--+        |              | <-----> |  Kernel |
|  | <----> |              |  Hcall  |         |
+--+  DRC2  +--------------+         +---------+
```

PAPR hypervisor terms shared hardware resources like PCI devices, NVDIMMs etc available for use by LPARs as Dynamic Resource (DR). When a DR is allocated to an LPAR, PHYP creates a data-structure called Dynamic Resource Connector (DRC) to manage LPAR access. An LPAR refers to a DRC via an opaque 32-bit number called DRC-Index. The DRC-index value is provided to the LPAR via device-tree where its present as an attribute in the device tree node associated with the DR.

HCALL Return-values

After servicing the hcall, hypervisor sets the return-value in *r3* indicating success or failure of the hcall. In case of a failure an error code indicates the cause for error. These codes are defined and documented in arch specific header [4].

In some cases a hcall can potentially take a long time and need to be issued multiple times in order to be completely serviced. These hcalls will usually accept an opaque value *continue-token* within there argument list and a return value of *H_CONTINUE* indicates that hypervisor hasn't still finished servicing the hcall yet.

To make such hcalls the guest need to set *continue-token == 0* for the initial call and use the hypervisor returned value of *continue-token* for each subsequent hcall until hypervisor returns a non *H_CONTINUE* return value.

HCALL Op-codes

Below is a partial list of HCALLs that are supported by PHYP. For the corresponding opcode values please look into the arch specific header [4]:

H_SCM_READ_METADATA

Input: *drcIndex, offset, buffer-address, numBytesToRead*
Out: *numBytesRead*
Return Value: *H_Success, H_Parameter, H_P2, H_P3, H_Hardware*

Given a DRC Index of an NVDIMM, read N-bytes from the metadata area associated with it, at a specified offset and copy it to provided buffer. The metadata area stores configuration information such as label information, bad-blocks etc. The metadata area is located out-of-band of NVDIMM storage area hence a separate access semantics is provided.

H_SCM_WRITE_METADATA

Input: *drcIndex, offset, data, numBytesToWrite*
Out: *None*
Return Value: *H_Success, H_Parameter, H_P2, H_P4, H_Hardware*

Given a DRC Index of an NVDIMM, write N-bytes to the metadata area associated with it, at the specified offset and from the provided buffer.

H_SCM_BIND_MEM

Input: *drcIndex, startingScmBlockIndex, numScmBlocksToBind, targetLogicalMemoryAddress, continue-token*
Out: *continue-token, targetLogicalMemoryAddress, numScmBlocksToBound*
Return Value: *H_Success, H_Parameter, H_P2, H_P3, H_P4, H_Overlap, H_Too_Big, H_P5, H_Busy*

Given a DRC-Index of an NVDIMM, map a continuous SCM blocks range *(startingScmBlockIndex, startingScmBlockIndex+numScmBlocksToBind)* to the guest at *targetLogicalMemoryAddress* within guest physical address space. In case *targetLogicalMemoryAddress == 0xFFFFFFFF_FFFFFFFF* then hypervisor assigns a target address to the guest. The HCALL can fail if the Guest has an active PTE entry to the SCM block being bound.

H_SCM_UNBIND_MEM | Input: drcIndex, startingScmLogicalMemoryAddress, numScmBlocksToUnbind | Out: numScmBlocksUnbound | Return Value: *H_Success, H_Parameter, H_P2, H_P3, H_In_Use, H_Overlap,* | *H_Busy, H_LongBusyOrder1mSec, H_LongBusyOrder10mSec*

Given a DRC-Index of an NVDimm, unmap *numScmBlocksToUnbind* SCM blocks starting at *startingScmLogicalMemoryAddress* from guest physical address space. The HCALL can fail if the Guest has an active PTE entry to the SCM block being unbound.

H_SCM_QUERY_BLOCK_MEM_BINDING

Input: *drcIndex, scmBlockIndex*
Out: *Guest-Physical-Address*
Return Value: *H_Success, H_Parameter, H_P2, H_NotFound*

Given a DRC-Index and an SCM Block index return the guest physical address to which the SCM block is mapped to.

H_SCM_QUERY_LOGICAL_MEM_BINDING

Input: *Guest-Physical-Address*

Out: *drcIndex, scmBlockIndex*
Return Value: *H_Success, H_Parameter, H_P2, H_NotFound*

Given a guest physical address return which DRC Index and SCM block is mapped to that address.

H_SCM_UNBIND_ALL

Input: *scmTargetScope, drcIndex*
Out: *None*
Return Value: *H_Success, H_Parameter, H_P2, H_P3, H_In_Use, H_Busy, H_LongBusyOrder1mSec, H_LongBusyOrder10mSec*

Depending on the Target scope unmap all SCM blocks belonging to all NVDIMMs or all SCM blocks belonging to a single NVDIMM identified by its drcIndex from the LPAR memory.

H_SCM_HEALTH

Input: drcIndex
Out: *health-bitmap (r4), health-bit-valid-bitmap (r5)*
Return Value: *H_Success, H_Parameter, H_Hardware*

Given a DRC Index return the info on predictive failure and overall health of the PMEM device. The asserted bits in the health-bitmap indicate one or more states (described in table below) of the PMEM device and health-bit-valid-bitmap indicate which bits in health-bitmap are valid. The bits are reported in reverse bit ordering for example a value of 0xC400000000000000 indicates bits 0, 1, and 5 are valid.

Health Bitmap Flags:

Bit	Definition
00	PMEM device is unable to persist memory contents. If the system is powered down, nothing will be saved.
01	PMEM device failed to persist memory contents. Either contents were not saved successfully on power down or were not restored properly on power up.
02	PMEM device contents are persisted from previous IPL. The data from the last boot were successfully restored.
03	PMEM device contents are not persisted from previous IPL. There was no data to restore from the last boot.
04	PMEM device memory life remaining is critically low
05	PMEM device will be garded off next IPL due to failure
06	PMEM device contents cannot persist due to current platform health status. A hardware failure may prevent data from being saved or restored.
07	PMEM device is unable to persist memory contents in certain conditions
08	PMEM device is encrypted
09	PMEM device has successfully completed a requested erase or secure erase procedure.
10:63	Reserved / Unused

H_SCM_PERFORMANCE_STATS

Input: drcIndex, resultBuffer Addr
Out: None
Return Value: *H_Success, H_Parameter, H_Unsupported, H_Hardware, H_Authority, H_Privilege*

Given a DRC Index collect the performance statistics for NVDIMM and copy them to the resultBuffer.

H_SCM_FLUSH

Input: *drcIndex, continue-token*
Out: *continue-token*
Return Value: *H_SUCCESS, H_Parameter, H_P2, H_BUSY*

Given a DRC Index Flush the data to backend NVDIMM device.

The hcall returns H_BUSY when the flush takes longer time and the hcall needs to be issued multiple times in order to be completely serviced. The *continue-token* from the output to be passed in the argument list of subsequent hcalls to the hypervisor until the hcall is completely serviced at which point H_SUCCESS or other error is returned by the hypervisor.

References

[1] "Power Architecture Platform Reference" https://en.wikipedia.org/wiki/Power_Architecture_Platform_Reference

[2] (*1*,*2*) "Linux on Power Architecture Platform Reference" https://members.openpowerfoundation.org/document/dl/469

[3] "Definitions and Notation" Book III-Section 14.5.3 https://openpowerfoundation.org/?resource_lib=power-isa-version-3-0

[4] (*1*,*2*,*3*) arch/powerpc/include/asm/hvcall.h

[5] "64-Bit ELF V2 ABI Specification: Power Architecture" https://openpowerfoundation.org/?resource_lib=64-bit-elf-v2-abi-specification-power-architecture