Platform Runtime Mechanism Proof-Of-Concept Codes

1. Background

Platform Runtime Mechanism (PRM) is an architecture for ACPI codes to call into UEFI BIOS's Runtime Services at OS runtime. Traditionally, ACPI codes call into BIOS through the invocation of Software SMIs. PRM provides an alternative for ACPI codes to invoke UEFI Runtime Services, which runs in OS kernel space, without invoking SMM.

This package contains proof-of-concept codes that implement the ideas of PRM.

IMPORTANT NOTE: The codes in this package are for proof-of-concept only. The codes do not represent a formal design, nor do the codes aim at product quality, or being used in products.

2. Overall Architecture

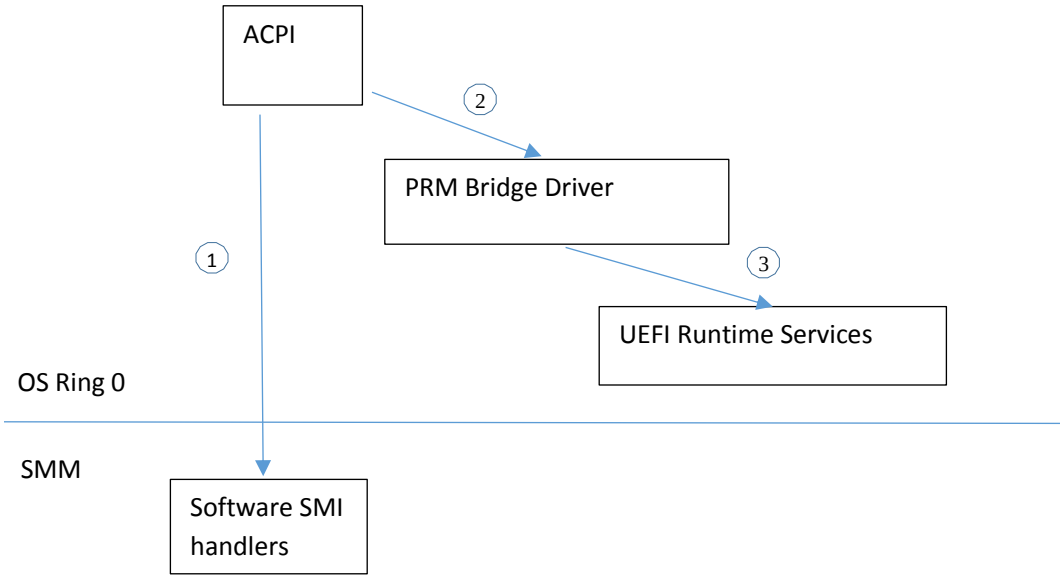

Figure 1: Overall Architecture

Figure 1 is explained as below:

①: The traditional way: ACPI calls software SMI

② ③: The PRM way: ACPI call is captured by a PRM Bridge Driver, and the PRM Bridge Driver calls into UEFI Runtime Services on behalf of the ACPI call, and returns the result to ACPI.

3. Code Contents

This package provides proof-of-concept codes for:

- A UEFI Runtime driver that exposes a Runtime function;
- ASL codes that define an object which the PRM Bridge Driver can bind to.

(The design of PRM Bridge driver is explained in section 4)

3.1 RtSample

This is the UEFI Runtime Driver that produces the PRM handler. This driver provides a GetVariable() wrapper that hooks the UEFI GetVariable Runtime service – if the GetVariable() wrapper receives a specific

GET_VAR_HOOK_GUID, it will pass control to the PRM handler; otherwise, it will pass control to the normal GetVariable() service.

This driver also illustrates how to use a passed in OS kernel service (e.g., the printk() function to output a debug message) in PRM handler.

Some common techniques used by a UEFI Runtime driver are shown too, such as setting RUNTIME attributes to the MMIO range that needs to be accessed, and changing addresses from BIOS stage 1:1 mapping address to OS virtual address at Virtual Address Change event. Please refer to the sample codes for details.

3.2 RuntimeTest.asl

This is the sample ASL code that defines a PRM ACPI Device ("PRMD"). It contains a method "PRMM" that notifies itself. Calling "PRMM" from other ACPI codes or from a device driver will trigger the PRM Bridge Driver's callback function registered to "PRMD"'s notifications.

The sample ASL code also shows an example to notify "PRMD" at an SCI event, where PRM handler can be used to handle asynchronous events.

4. PRM Bridge Driver

The PRM Bridge Driver is an OS driver that does following work:

- Binds itself to the PRM ACPI Device.
- Registers a callback function to the Notification of the PRM ACPI Device.
    - This callback function is invoked by OS whenever the PRM ACPI Device is notified
- In the Notification callback function:
    - Calls the GetVariable() UEFI Runtime Service, with the GET_VAR_HOOK_GUID, to invoke the PRM handler. The Data pointer of the GetVariable() call points to a Parameter Structure, which optionally contains:
        - A pointer to OS services that can be used by PRM handler, e.g., a debug print function for PRM handler to print debug messages
        - Parameters of this ACPI call (extracted from a predefined ACPI OpRegion)
        - (Sample codes in RtSample only shows how to use a pointer to OS's debug print function)
    - Return the result of GetVariable() UEFI Runtime call by placing the results into the predefined ACPI OpRegion. (ACPI codes have the option to handle the results accordingly, e.g., perform some tasks and make further PRM calls, if needed.)

To implement PRM Bridge Driver on Linux, please

- Refer to Linux Kernel's drivers\acpi\acpi_video.c on how a driver registers itself to an ACPI device and gets notified on the ACPI device's notification;
- Refer to Linux Kernel's drivers\infiniband\hw\hfi1\efivar.c on how to call UEFI GetVariable() service

5. Conclusion

These proof-of-concept codes shows BIOS can provide services to ACPI, not only from SMM, but also from OS ring 0 space where applicable.

PRM can only be used in cases where there is no privileged SMM hardware operations involved, and where all work can be done in Ring 0 without security risks. Keep in mind PRM handler runs with the same level of security as OS drivers and normal UEFI Runtime Services.

References:

- PRM introduction: https://www.opencompute.org/events/past-summits (Please search for "UEFI Implementation Intel® Xeon Based OCP Platform" in the webpage, which lists pointers to video and presentation that introduce the concepts of PRM and SMM reduction.)
- PRM Case Study: https://www.opencompute.org/events/past-summits (Please search for "Case Study: Alternatives for SMM usage in Intel Platforms" in the webpage, which lists pointers to video and presentation that introduce a case study for SMM reduction.)
- TianoCore: https://www.tianocore.org/
- UEFI Forum: https://uefi.org/