ACPI _OSI and _REV methods

An ACPI BIOS can use the "Operating System Interfaces" method (_OSI) to find out what the operating system supports. Eg. If BIOS AML code includes _OSI("XYZ"), the kernel's AML interpreter can evaluate that method, look to see if it supports 'XYZ' and answer YES or NO to the BIOS.

The ACPI _REV method returns the "Revision of the ACPI specification that OSPM supports"

This document explains how and why the BIOS and Linux should use these methods. It also explains how and why they are widely misused.

How to use _OSI

Linux runs on two groups of machines -- those that are tested by the OEM to be compatible with Linux, and those that were never tested with Linux, but where Linux was installed to replace the original OS (Windows or OSX).

The larger group is the systems tested to run only Windows. Not only that, but many were tested to run with just one specific version of Windows. So even though the BIOS may use _OSI to query what version of Windows is running, only a single path through the BIOS has actually been tested. Experience shows that taking untested paths through the BIOS exposes Linux to an entire category of BIOS bugs. For this reason, Linux _OSI defaults must continue to claim compatibility with all versions of Windows.

But Linux isn't actually compatible with Windows, and the Linux community has also been hurt with regressions when Linux adds the latest version of Windows to its list of _OSI strings. So it is possible that additional strings will be more thoroughly vetted before shipping upstream in the future. But it is likely that they will all eventually be added.

What should an OEM do if they want to support Linux and Windows using the same BIOS image? Often they need to do something different for Linux to deal with how Linux is different from Windows. Here the BIOS should ask exactly what it wants to know:

_OSI("Linux-OEM-my_interface_name") where 'OEM' is needed if this is an OEM-specific hook, and 'my_interface_name' describes the hook, which could be a quirk, a bug, or a bug-fix.

In addition, the OEM should send a patch to upstream Linux via the linux-acpi@vger.kernel.org mailing list. When that patch is checked into Linux, the OS will answer "YES" when the BIOS on the OEM's system uses _OSI to ask if the interface is supported by the OS. Linux distributors can back-port that patch for Linux pre-installs, and it will be included by all distributions that re-base to upstream. If the distribution can not update the kernel binary, they can also add an acpi_osi=Linux-OEM-my_interface_name cmdline parameter to the boot loader, as needed.

If the string refers to a feature where the upstream kernel eventually grows support, a patch should be sent to remove the string when that support is added to the kernel.

That was easy. Read on, to find out how to do it wrong.

Before _OSI, there was _OS

ACPI 1.0 specified "_OS" as an "object that evaluates to a string that identifies the operating system."

The ACPI BIOS flow would include an evaluation of _OS, and the AML interpreter in the kernel would return to it a string identifying the OS:

Windows 98, SE: "Microsoft Windows" Windows ME: "Microsoft WindowsME:Millennium Edition" Windows NT: "Microsoft Windows NT"

The idea was on a platform tasked with running multiple OS's, the BIOS could use _OS to enable devices that an OS might support, or enable quirks or bug workarounds necessary to make the platform compatible with that pre-existing OS.

But _OS had fundamental problems. First, the BIOS needed to know the name of every possible version of the OS that would run on it, and needed to know all the quirks of those OS's. Certainly it would make more sense for the BIOS to ask *specific* things of the OS, such "do you support a specific interface", and thus in ACPI 3.0, _OSI was born to replace _OS.

_OS was abandoned, though even today, many BIOS look for _OS "Microsoft Windows NT", though it seems somewhat far-fetched that anybody would install those old operating systems over what came with the machine.

Linux answers "Microsoft Windows NT" to please that BIOS idiom. That is the *only* viable strategy, as that is what modern Windows does, and so doing otherwise could steer the BIOS down an untested path.

_OSI is born, and immediately misused

With _OSI, the *BIOS* provides the string describing an interface, and asks the OS: "YES/NO, are you compatible with this interface?"

eg. _OSI("3.0 Thermal Model") would return TRUE if the OS knows how to deal with the thermal extensions made to the ACPI 3.0 specification. An old OS that doesn't know about those extensions would answer FALSE, and a new OS may be able to return

TRUE.

For an OS-specific interface, the ACPI spec said that the BIOS and the OS were to agree on a string of the form such as "Windows-interface_name".

But two bad things happened. First, the Windows ecosystem used _OSI not as designed, but as a direct replacement for _OS -- identifying the OS version, rather than an OS supported interface. Indeed, right from the start, the ACPI 3.0 spec itself codified this misuse in example code using _OSI("Windows 2001").

This misuse was adopted and continues today.

Linux had no choice but to also return TRUE to _OSI("Windows 2001") and its successors. To do otherwise would virtually guarantee breaking a BIOS that has been tested only with that _OSI returning TRUE.

This strategy is problematic, as Linux is never completely compatible with the latest version of Windows, and sometimes it takes more than a year to iron out incompatibilities.

Not to be out-done, the Linux community made things worse by returning TRUE to _OSI("Linux"). Doing so is even worse than the Windows misuse of _OSI, as "Linux" does not even contain any version information. _OSI("Linux") led to some BIOS' malfunctioning due to BIOS writer's using it in untested BIOS flows. But some OEM's used _OSI("Linux") in tested flows to support real Linux features. In 2009, Linux removed _OSI("Linux"), and added a cmdline parameter to restore it for legacy systems still needed it. Further a BIOS_BUG warning prints for all BIOS's that invoke it.

No BIOS should use _OSI("Linux").

The result is a strategy for Linux to maximize compatibility with ACPI BIOS that are tested on Windows machines. There is a real risk of over-stating that compatibility; but the alternative has often been catastrophic failure resulting from the BIOS taking paths that were never validated under *any* OS.

Do not use _REV

Since _OSI("Linux") went away, some BIOS writers used _REV to support Linux and Windows differences in the same BIOS.

_REV was defined in ACPI 1.0 to return the version of ACPI supported by the OS and the OS AML interpreter.

Modern Windows returns _REV = 2. Linux used ACPI_CA_SUPPORT_LEVEL, which would increment, based on the version of the spec supported.

Unfortunately, _REV was also misused. eg. some BIOS would check for _REV = 3, and do something for Linux, but when Linux returned _REV = 4, that support broke.

In response to this problem, Linux returns _REV = 2 always, from mid-2015 onward. The ACPI specification will also be updated to reflect that _REV is deprecated, and always returns 2.

Apple Mac and _OSI("Darwin")

On Apple's Mac platforms, the ACPI BIOS invokes _OSI("Darwin") to determine if the machine is running Apple OSX.

Like Linux's _OSI("*Windows*") strategy, Linux defaults to answering YES to _OSI("Darwin") to enable full access to the hardware and validated BIOS paths seen by OSX. Just like on Windows-tested platforms, this strategy has risks.

Starting in Linux-3.18, the kernel answered YES to _OSI("Darwin") for the purpose of enabling Mac Thunderbolt support. Further, if the kernel noticed _OSI("Darwin") being invoked, it additionally disabled all _OSI("*Windows*") to keep poorly written Mac BIOS from going down untested combinations of paths.

The Linux-3.18 change in default caused power regressions on Mac laptops, and the 3.18 implementation did not allow changing the default via cmdline "acpi_osi=!Darwin". Linux-4.7 fixed the ability to use acpi_osi=!Darwin as a workaround, and we hope to see Mac Thunderbolt power management support in Linux-4.11.