Proceedings of the Linux Symposium

June 27th–30th, 2007
Ottawa, Ontario
Canada

ACPI in Linux® – Myths vs. Reality

Len Brown
Intel Open Source Technology Center
len.brown@intel.com

Abstract

Major Linux distributors have been shipping ACPI in Linux for several years, yet mis-perceptions about ACPI persist in the Linux community. This paper addresses the most common myths about ACPI in Linux.

1 Myth: There is no benefit to enabling ACPI on my notebook, desktop, or server.

When users boot in ACPI-mode instead of legacy-mode, the first thing they notice is that the power button is now under software control. In legacy-mode, it is a physical switch which immediately removes power. In ACPI mode, the button interrupts the OS, which can shutdown gracefully. Indeed, ACPI standardizes the power, sleep, and lid buttons and the OS can map them to whatever actions it likes.

In addition to standardizing power button events, ACPI also standardizes how the OS invokes software controlled power-off. So a software-initiated power off removes power from the system after Linux has shutdown, while on many systems in legacy-mode, the operator must manually remove power.

Users notice improved battery life when running in ACPI-mode. On today's notebooks, a key contributor to improved battery life is processor power management. When the processor is idle, the Linux idle loop takes advantage of ACPI CPU idle power states (C-states) to save power. When the processor is partially idle, the Linux cpufreq sub-system takes advantage of ACPI processor performance states (P-states) to run the processor at reduced frequency and voltage to save power.

Users may notice other differences depending on the platform and the GUI they use, such as battery capacity alarms, thermal control, or the addition of or changes in the the ability to invoke suspend-to-disk or suspend-to-RAM, etc.

Users with an Intel® processor supporting Hyper-Threaded Technology (HT) will notice that HT is enabled in ACPI-mode, and not available in legacy-mode.

Many systems today are multi-processor and thus use an IO-APIC to direct multiple interrupt sources to multiple processors. However, they often do not include legacy MPS (Multi-Processor Specification) support. Thus, ACPI is the only way to configure the IO-APIC on these systems, and they'll run in XT-PIC mode when booted without ACPI.

But to look at ACPI-mode vs. legacy-mode a bit deeper, it is necessary to look at the specifications that ACPI replaces. The most important one is Advanced Power Management [APM], but ACPI also obsoletes the Multi-Processor Specification [MPS] and the PCI IRQ Routing Specification [PIRQ].

1.1 ACPI vs. Advanced Power Management (APM)

APM and ACPI are mutually exclusive. While a flexible OS can include support for both, the OS can not enable both on the same platform at the same time.

APM 1.0 was published in 1992 and was supported by Microsoft® Windows® 3.1. The final update to APM, 1.2, was published in 1996.

ACPI 1.0 was developed in 1996, and Microsoft first added support for it in Windows NT®. For a period, Windows preferred ACPI, but retained APM support to handle older platforms. With the release of Windows Vista™, Microsoft has completed the transition to ACPI by dropping support for APM.

Many platform vendors deleted their APM support during this transition period, and few will retain APM support now that this transition is complete.

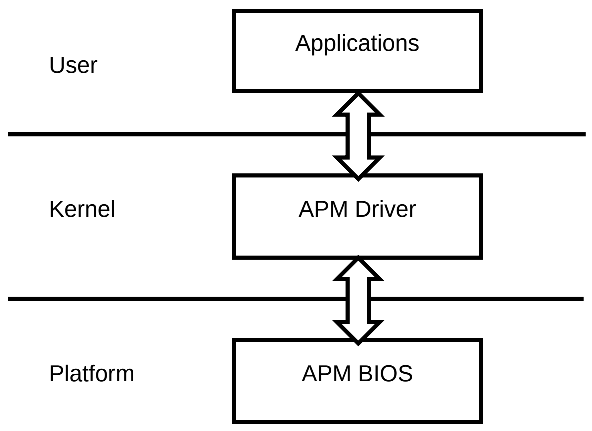

Figure 1: APM Architecture

1.1.1 APM overview

The goal of the APM specification was to extend battery life, while hiding the details of how that is done from the OS in the APM BIOS.

APM defines five general system power states: Full On, APM Enabled, APM Standby, APM Suspend, and Off. The Full On state has no power management. The APM Enabled state may disable some unused devices. The APM Standby state was intended to be a system state with instantaneous wakeup latency. The APM Suspend was optionally suspend-to-RAM, and/or hibernate-to-disk.

APM defines analogous power states for devices: Device On, Device Power Managed, Device Lower Power, and Device Off. Device context is lost in the Device Off state. APM is somewhat vague about whether it is the job of the OS device driver or the APM BIOS to save and restore the device-operational parameters around Device Off.

APM defines CPU Core control states—Full On, Slow Clock, and Off. Interrupts transition the CPU back to Full On instantaneously.

An APM-aware OS has an APM Driver that connects with the APM BIOS. After a connection is established, the APM Driver and APM BIOS "cooperatively perform power management." What this means is that the OS makes calls into the BIOS to discover and modify the default policies of the APM BIOS, and the OS polls the BIOS at least once per second for APM BIOS events.

The APM BIOS can report events to the APM Driver. For example, after detecting an idle period, the APM BIOS may issue a System Standby Request Notification telling the OS that it wants to suspend. The OS must answer by calling a Set Power State function within a certain time. If the OS doesn't answer within the appropriate time, the BIOS may suspend the system anyway. On resume, APM issues a System Standby Resume Notification to let the OS know what happened. This is the OS's opportunity to update its concept of time-of-day.

The OS can disable APM's built-in concept of requesting a suspend or standby, and can instead manually ask APM to perform these transitions on demand.

The OS can instrument its idle loop to call into the APM BIOS to let it know that the processor is idle. The APM BIOS would then perhaps throttle the CPU if it appeared to be running faster than necessary.

The APM BIOS knows about AC/DC status. The APM Driver can query the BIOS for current status, and can also poll for AC/DC change events.

The APM BIOS knows about battery topology and status. The APM Driver can query for configuration as well as capacity, and can poll for Battery Low Notification.

APM supports a hard-coded list of devices for power management including display, disk, parallel ports, serial ports, network adapters, and PCMCIA sockets. The OS can query for their state, enable/disable the APM BIOS from managing the devices, and poll for state changes.

1.1.2 Why APM is not viable

APM is fundamentally centered around the the APM BIOS. The APM BIOS is entered from OS APM Driver calls as well as from System Management Interrupts (SMI) into System Management Mode (SMM). SMM is necessary to implement parts of APM since BIOS code needs to run on the processor without the knowledge or support of the OS.

But it turns out that calling into the BIOS is a really scary thing for an operating system to do. The OS puts the stability of the system in the hands of the BIOS developer on every call. Indeed, the only thing more frightening to the OS is SMM itself, which is completely transparent to the OS and thus virtually impossible to debug. The largest problem with both of these is that if the state of the system was not as the BIOS developer expected it, then it may not be restored properly on BIOS return to the OS.

So the quality of the "APM experience" varied between platforms depending on the platform's APM BIOS implementation.

Further, the APM specification includes hard-coded limitations about the system device configuration. It is not extensible such that the platform firmware can accommodate system configurations that did not exist when the specification was written.

The philosophy of ACPI, on the other hand, is to put the OS, not the BIOS, in charge of power management policy. ACPI calls this OSPM, or "Operating System-directed configuration and Power Management." OSPM never jumps into the BIOS in ACPI-mode. However, it does access system devices and memory by interpreting BIOS ACPI Machine Language (AML) in kernel mode.

ACPI reduces the the need for SMM, but SMM is still a tool available to BIOS writers to use when they see fit.

ACPI's AML is extensible. It can describe resources and capabilities for devices that the specification knows nothing about—giving the OS the ability to configure and power-manage a broad range of system configurations over time.

ACPI 1.0 was published at the end of 1996. It is probably fair to say that platforms did not universally deploy it until ACPI 2.0 was published in 2000. At that time, Microsoft released Windows 2000, and the APM era was effectively over.

So if you've got a notebook from 1998 or 1999 that includes both APM and ACPI support, you may find that its APM implementation is more mature (and better tested) than its new ACPI implementation. Indeed, it is true that the upstream Linux kernel enables ACPI on all systems that advertise ACPI support, but I recommend that Linux distributors ship with `CONFIG_ACPI_BLACKLIST_YEAR=2000` to disable ACPI in Linux on machines from the last century.

1.2 Multi-Processor Specification (MPS)

MPS 1.1 was issued in 1994. The latest revision, MPS 1.4, was issued in 1995, with minor updates until May, 1997. The primary goal of MPS was to standardize multi-processor configurations such that a "shrink-wrap" OS could boot and run on them without customization. Thus a customer who purchased an MPS-compliant system would have a choice of available Operating Systems.

MPS mandated that the system be symmetric—all processors, memory, and I/O are created equal. It also mandated the presence of Local and I/O APICs.

The Local APIC specified support for inter-processor interrupts—in particular, the INIT IPI and STARTUP IPI used to bring processors on-line.

While the specification calls it an "MP BIOS," the code is much different from the "APM BIOS." The MP BIOS simply puts all the processors into a known state so that the OS can bring them on-line, and constructs static MP configuration data structures—the MP tables—that enumerate the processors and APICS for the OS.

MPS also specified a standard memory map. However, this memory map was later replaced by e820, which is part of the ACPI specification.

The MPS tables enumerate processors, buses, and IO-APICs; and the tables map interrupt sources to IO-APIC input pins.

MPS mandated that SMP siblings be of equal capability. But when Intel introduced Hyper-Threading Technology (HT) with the Pentium® 4 processor, suddenly siblings where not all created equal. What would happen to the installed base if MPS enumerated HT siblings like SMP siblings? Certainly if an HT-ignorant OS treated HT siblings as SMP, it would not schedule tasks optimally.

So the existing MPS 1.4 was not extended to handle HT,[1] and today HT siblings are only available to the OS by parsing the ACPI tables.

But MPS had a sister specification to help machines handle mapping interrupt wires in PIC an IO-APIC mode—the PIRQ spec.

1.3 ACPI vs. PCI IRQ Routers (PIRQ)

IRQ routers are motherboard logic devices that connect physical IRQ wires to different interrupt controller in-

[1]Some BIOSes have a SETUP option to enumerate HT siblings in MPS, but this is a non-standard feature.

put pins. Microsoft published [PIRQ], describing OS-visible tables for PIRQ routers. However, the spec excludes any mention of a standard method to get and set the routers—instead stating that Microsoft will work with the chipset vendors to make sure Windows works with their chipsets.

ACPI generalizes PIRQ routers into ACPI PCI Interrupt Link Devices. In real-life, these are just AML wrappers for both the contents of the PIRQ tables, and the undocumented chipset-specific get/set methods above. The key is that the OS doesn't need any chipset-specific knowledge to figure out what links can be set to, what they are currently set to, or to change the settings. What this means is that the ACPI-aware OS is able to route interrupts on platforms for which it doesn't have intimate knowledge.

1.4 With benefits come risks

It is fair to say that the Linux/ACPI sub-system is large and complex, particularly when compared with BIOS-based implementations such as APM. It is also fair to say that enabling ACPI—effectively an entire suite of features—carries with it a risk of bugs. Indeed it carries with it a risk of regressions, particularly on pre-2000 systems which may have a mature APM implementation and an immature ACPI implementation.

But the hardware industry has effectively abandoned the previous standards that are replaced by ACPI. Further, Linux distributors are now universally shipping and supporting ACPI in Linux. So it is critical that the Linux community continue to build the most robust and full featured ACPI implementation possible to benefit its users.

2 Myth: Suspend to Disk doesn't work, it must be ACPI's fault.

The suspend-to-disk (STD) implementation in Linux has very little to do with ACPI. Indeed, if STD is not working on your system, try it with `acpi=off` or `CONFIG_ACPI=n`. Only if it works better without ACPI can you assume that it is an ACPI-specific issue.

ACPI's role during suspend-to-disk is very much like its role in a normal system boot and a normal system power-off. The main difference is that when ACPI is available, STD uses the "platform" method to power off the machine instead of the "shutdown" method. This allows more devices to be enabled as wakeup devices, as some can wake the system from suspend to disk, but not from soft power-off.

Many end-users think think that STD and ACPI are practically synonyms. One reason for this is because in the past, `/proc/acpi/sleep` was used to invoke STD. However, this method is now deprecated in favor of the generic `/sys/power/state` interface.

Note that suspend-to-RAM (STR) is more dependent on ACPI than STD, as the sleep and wakeup paths are ACPI-specific.

However, the vast majority of both STD and STR failures today have nothing to do with ACPI itself, but instead are related to device drivers. You can often isolate these issues by unloading a device driver before suspend, and re-loading it after resume.

3 Myth: The buttons on my notebook don't work, it must be ACPI's fault.

The ACPI specification standardizes only 3 buttons—power, sleep, and lid. If these buttons do not work in ACPI-mode, then it is, indeed, an ACPI issue.

The other buttons on the keyboard are handled in a variety of platform-specific methods.

First there are the standard keyboard buttons that go directly to the input sub-system. When these malfunction, it cannot be blamed on ACPI, because if there is a problem with them, they'll have the same problem with `acpi=off`.

The "ACPI issues" appear with "hot-keys," which are platform-specific buttons that are not handled by the standard keyboard driver.

When in `acpi=off` mode, these are sometimes handled in the BIOS with SMI/SMM. But when in ACPI-mode, this SMM support is disabled by the platform vendor on the assumption that if ACPI-mode is enabled, then a modern OS sporting a platform-specific hot-key driver is available to handle the hot-keys. For Windows, the vendor may be able to guarantee this is true. However, to date, no platform vendor has volunteered to create or support the community in the creation of platform-specific hot-key drivers for Linux.

Sometimes these keys actually do use the ACPI subsystem to get their work done. However, they report events to vendor-specific ACPI devices, which need vendor-specific ACPI device drivers to receive the events and map them to actions.

4 Myth: My motherboard boots with `acpi=off`, but fails otherwise, it must be ACPI's fault.

With the advent of multi-core processors, SMP systems are very common, and all modern x86 SMP system have an IO-APIC.

However, many notebook and desktop BIOS vendors do not include MPS support in their BIOS. So when booted with `acpi=off`, these systems revert all the way back to 8259 PIC mode. So only in ACPI-mode is the IO-APIC enabled, and thus any IO-APIC mode issues get blamed on ACPI.

The real ACPI vs. non-ACPI, apples-versus-apples comparison would be `acpi=off` vs. `noapic`—for both of these will boot in PIC mode.

But why do so many systems have trouble with IO-APIC mode? The most common reason is the periodic HZ timer. Linux typically uses the 8254 Programmable Interrupt Timer (PIT) for clock ticks. This timer is typically routed to IRQ0 via either IO-APIC pin0 or pin2.

But Windows doesn't always use the PIT; it uses the RTC on IRQ8. So a system vendor that validates their system only with Windows and never even boots Linux before releasing their hardware may not notice that the 8254 PIT used by Linux is not hooked up properly.

5 Myth: The Linux community has no influence on the ACPI Specification.

HP, Intel, Microsoft, Phoenix, and Toshiba co-developed the ACPI specification in the mid-1990s, but it continues to evolve over time. Indeed, version 3.0b was published in October, 2006.

Linux plays a role in that evolution. The latest version of the specification included a number of "clarifications" that were due to direct feedback from the Linux community.

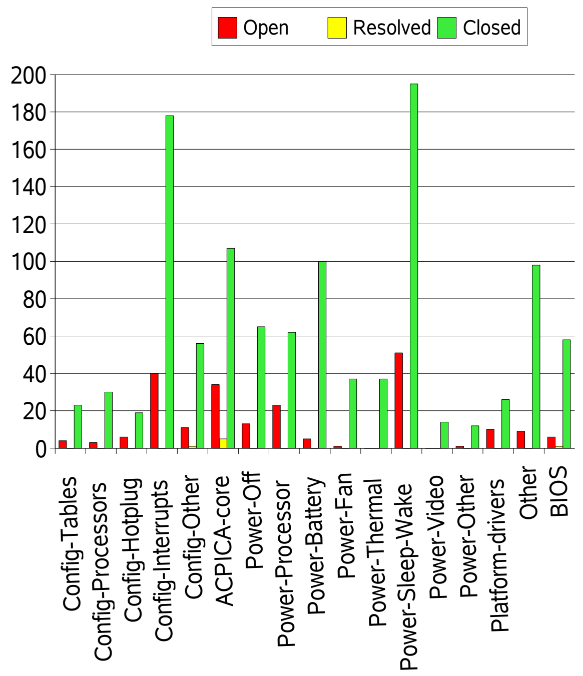

Figure 2: ACPI sighting profile at bugzilla.kernel.org

Sometimes Linux fails to work properly on a system in the field and the root cause is that the ACPI specification was vague. This caused the Linux implementation to do one thing, while the BIOS vendors and Microsoft did another thing.

The Linux/ACPI team in the Intel Open Source Technology Center submits "specification clarifications" directly to the ACPI committee when this happens. The specification is updated, and Linux is changed to match the corrected specification.

6 Myth: ACPI bugs are all due to sub-standard platform BIOS.

When Linux implemented and shipped ACPI, we ran into three categories of failures:

1. Linux fails because the platform BIOS clearly violates the written ACPI specification.

   These failures exist because until Linux implemented ACPI, platform vendors had only Windows OS compatibility tests to verify if their ACPI implementation was correct.

Unfortunately, an OS compatibility test is not a specification compliance test. The result is that many BIOS implementations work by accident because they have been exercised by only one OS.

Today, the Linux-ready Firmware Developer Kit [FWKIT] is available so that vendors who care about Linux have the tools they need to assure their BIOS implementation is compatible with Linux.

2. Linux fails because the platform BIOS writer and Linux programmer interpreted the ACPI specification differently.

   As mentioned in the previous section, we treat these as Linux bugs, fix them, and update the specification to match the actual industry practice.

3. Bugs in the Linux/ACPI implementation. These are simply bugs in Linux, like any other bugs in Linux.

The myth is that a large number of failures are due BIOS bugs in category #1. The reality is shown by Figure 2—under 10% of all Linux/ACPI bugs can be blamed on broken BIOSes.

The majority of bugs have actually been reported against category #3, the Linux-specific code.

7 Myth: ACPI code seems to change a lot, but isn't getting any better.

When ACPI was still new in Linux and few distributors were shipping it, there were many times when changes would fix several machines, but break several others. To be honest, a certain amount of experimentation was going on to figure out how to become bug-compatible with the installed base of systems.

Marcelo Tosatti was maintaining Linux-2.4, and he walked up to me and in a concerned voice asked why we'd break some systems while fixing others. It was clear we needed validation tests to prevent regressions, but we didn't have them yet. And before we did, Linux distributors cut over to Linux-2.6, and almost universally started shipping ACPI. Suddenly we had a large installed base running Linux/ACPI.

For a short while we didn't mend our ways of experimenting on the user base. Then Linus Torvalds scolded us for knowingly causing regressions, insisting that even if a system is working by mistake, changes should never knowingly break the installed base. He was right, of course, and ever since the Linux/ACPI team has made regressions the highest-priority issues.

But while this was happening, a team at Intel was creating three test suites that today are used to to verify that Linux/ACPI is constantly improving.

1. The ACPICA ASL Test Suite (ASLTS) is distributed in open source along with the ACPICA source package on `intel.com`. [ACPICA] ASLTS runs a suite of over 2,000 ASL tests against the ACPICA AML interpreter in a simulation environment. This is the same interpreter that resides in the Linux Kernel. During the development of this test suite, over 300 issues were found. Today there are fewer than 50 unresolved. ACPICA changes are not released if there are any regressions found by this test suite.

2. The ACPICA API Test Suite exercises the interfaces to the ACPICA core as seen from the OS. Like ASLTS, the API tests are done in a simulation environment.

3. ACPI ABAT—Automated Basic Acceptance Tests—which run on top of Linux, exercising user-visible features that are implemented by ACPI. ACPI ABAT is published in open source on the Linux/ACPI home page.[2]

Finally, one can examine the bug profile at `bugzilla.kernel.org` and observe that 80% of all sightings are now closed.

8 Myth: ACPI is slow and thus bad for high-performance cpufreq governors such as "ondemand."

It is true that the BIOS exports AML to the OS, which must use an AML interpreter to parse it. It is true that parsing AML is not intended to occur on performance-critical paths. So how can ACPI possibly be appropriate for enabling P-state transitions such as those made by the high-performance "ondemand" P-state governor—many times per second?

[2]`http://acpi.sourceforge.net`

The answer is that AML is used to parse the ACPI tables to tell cpufreq what states `ondemand` has to choose from. `ondemand` then implements its run-time policy without any involvement from ACPI.

The exception to this rule is that the platform may decide at run-time that the number of P-states should change. This is a relatively rare event, e.g. on an AC→DC transition, or a critical thermal condition. In this case, ACPI re-evaluates the list of P-states and informs cpufreq what new states are available. Cpufreq responds to this change and then proceeds to make its run-time governor decisions without any involvement from ACPI.

9 Myth: Speedstep-centrino is native and thus faster than ACPI-based 'acpi-cpufreq.'

To change the processor frequency and voltage, the OS can either write directly to native, model-specific registers (MSR), or it can access an IO address. There can be a significant efficiency penalty for IO access on some systems, particularly those which trap into SMM on that access.

So the community implemented speedstep-centrino, a cpufreq driver with hard-coded P-state tables based on CPUID and the knowledge of native MSRs. Speedstep-centrino did not need ACPI at all.

At that time, using acpi-cpufreq instead of speedstep-centrino meant using the less-efficient IO accesses. So the myth was true—but two things changed.

1. Speedstep-centrino's hard-coded P-state tables turned out to be difficult to maintain. So ACPI-table capability was added to speedstep-centrino where it would consult ACPI for the tables first, and use the hard-coded tables as a backup.

2. Intel published the "Intel Processor Vendor-Specific ACPI Interface Specification" along with [ACPICA]. This specification made public the bits necessary for an ACPI implementation to use native MSR access. So native MSR access was added to acpi-cpufreq.

The result was that both drivers could talk ACPI, and both could talk to MSRs. Speedstep-centrino still had its hard-coded tables, and acpi-cpufreq could still talk to IO addresses if the system asked it to.

Recently, acpi-cpufreq has been anointed the preferred driver of the two, and speedstep-centrino is scheduled for removal from the source tree as un-maintainable.

10 Myth: More CPU idle power states (C-states) are better than fewer states.

Users observe the system C-states in `/proc/acpi/processor/*/power` and assume that systems with more C-states save more power in idle than systems with fewer C-states. If they look at the data book for an Intel Core™ 2 Duo processor and try to relate those states to this file, then that is a reasonable assumption.

However, with some limitations, the mapping between hardware C-states and the ACPI C-states seen by Linux is arbitrary. The only things that matter with C-states is the amount of power saved, and the latency associated with waking up the processor. Some systems export lots of C-states, others export fewer C-states and have power-saving features implemented behind the scenes.

An example of this is Dynamic Cache Sizing. This is not under direct OS or C-state control. However, the processor recognizes that when deep C-states are entered, it can progressively flush more and more of the cache. When the system is very idle, the cache is completely flushed and is totally powered off. The user cannot tell if this feature is implemented in the processor by looking at how many C-states are exported to the OS—it is implemented behind the scenes in processor firmware.

11 Myth: Throttling the CPU will always use less energy and extend battery life.

$Energy = Power * Time$. That is to say, *[Watt-Hours] = [Watts] * [Hours]*.

Say the processor is throttled to half clock speed so that it runs at half power, but takes twice as long to get the job done. The energy consumed to retire the workload is the same and the only effect was to make the work take twice as long. Energy/work is constant.

There are, however, some second-order effects which make this myth partially true. For one, batteries are not ideal. They tend to supply more total energy when drained at a lower rate than when drained at a higher rate.

Secondly, systems with fans require energy to run those fans. If the system can retire the workload without getting hot, and succeeds in running the fans slower (or off), then less energy is required to retire the workload.

Note that processor clock throttling (ACPI T-states) discussed here should not be confused with processor performance states (ACPI P-states). P-states simultaneously reduce the voltage with the clock speed. As power varies as voltage squared, deeper P-states do take the processor to a more efficient energy/work operating point and minimize energy/work.

12 Myth: I can't contribute to improving ACPI in Linux.

The basis of this last myth may be the existence of ACPICA.

ACPICA (ACPI Component Architecture) is Intel's reference implementation of the ACPI interpreter and surrounding OS-agnostic code. In addition to Linux, BSD®, Solaris™, and other operating systems rely on ACPICA as the core of their ACPI implementation. For this to work, Intel holds the copyright on the code, and publishes under dual BSD and GPL licenses.

In Linux, 160 ACPICA files reside in sub-directories under `/drivers/acpi`. When a patch is submitted from the Linux community to those files, the Linux/ACPI maintainer asks for their permission to license the change to Intel to re-distribute under both licenses on the file, not just the GPL. That way, Intel can share the fix with the other ACPICA users rather than having the multiple copies diverge.

The ACPICA license isn't a barrier for open source contributors, but since it isn't straight GPL and extra permission is requested, it does generate a false impression that patches are unwelcome.

Further, it is the 40 pure-GPL files in `/drivers/acpi` that are most interesting to the majority of the Linux community anyway, for those files contain all the Linux-specific code and ACPI-related policy—treating the ACPICA core as a "black box."

But submitting patches is only one way to help. As described earlier, a lot of the work surrounding Linux/ACPI is determining what it means to be bug-compatible with common industry platform BIOS practice. The more people that are testing and poking at ACPI-related functions on a broad range of systems, the easier it is for the developers to improve the subsystem.

Your system should boot as well (or better) in ACPI-mode using no boot parameters as it does with `acpi=off` or other workarounds. Further, the power management features supported by ACPI such as suspend-to-RAM and processor power management should function properly and should never stop functioning properly.

It is a huge benefit to the community and the quality of the Linux ACPI implementation when users insist that their machines work properly—without the aid of workarounds. When users report regressions, file bugs, and test fixes, they are doing the community a great services that has a dramatic positive impact on the quality of ACPI in Linux.

13 Conclusion

Forget your initial impressions of Linux/ACPI made years ago. ACPI in Linux is not a myth, it is now universally deployed by the major Linux distributors, and it must function properly. Insist that the ACPI-related features on your system work perfectly. If they don't, complain loudly and persistently[3] to help the developers find and fix the issues.

The community must maintain its high standards for ACPI in Linux to continuously improve into the highest quality implementation possible.

References

[ACPI] Hewlett-Packard, Intel, Microsoft, Phoenix, Toshiba, *Advanced Configuration and Power Interface*, Revision 3.0b, October, 2006. `http://www.acpi.info`

[ACPICA] Intel, *ACPI Component Architecture*, `http://www.intel.com/technology/iapc/acpi/downloads.htm`

[APM] Intel, Microsoft *Advanced Power Management BIOS Interface Specification*, Revision 1.2, February, 1996. `http://www.microsoft.com/whdc/archive/amp_12.mspx`

[3]Start with `linux-acpi@vger.kernel.org` for Linux/ACPI related issues.

[FWKIT] *Linux-Ready Firmware Developer Kit*, `http://www.linuxfirmwarekit.org`

[MPS] Intel, *MultiProcessor Specification*, Version 1.4, May, 1997. `http://www.intel.com/design/intarch/MANUALS/242016.htm`

[PIRQ] Microsoft, *PCI IRQ Routing Table Specification*, Version 1.0, February, 1996. `http://www.microsoft.com/whdc/archive/pciirq.mspx`