Reprinted from the

Proceedings of the Linux Symposium

Volume One

July 21th–24th, 2004
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Review Committee

Jes Sorensen, *Wild Open Source, Inc.*
Matt Domsch, *Dell*
Gerrit Huizenga, *IBM*
Matthew Wilcox, *Hewlett-Packard*
Dirk Hohndel, *Intel*
Val Henson, *Sun Microsystems*
Jamal Hadi Salimi, *Znyx*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*

The State of ACPI in the Linux Kernel

A. Leonard Brown
Intel
len.brown@intel.com

Abstract

ACPI puts Linux in control of configuration and power management. It abstracts the platform BIOS and hardware so Linux and the platform can interoperate while evolving independently.

This paper starts with some background on the ACPI specification, followed by the state of ACPI deployment on Linux.

It describes the implementation architecture of ACPI on Linux, followed by details on the configuration and power management features.

It closes with a summary of ACPI bugzilla activity, and a list of what is next for ACPI in Linux.

1 ACPI Specification Background

> "ACPI (Advanced Configuration and Power Interface) is an open industry specification co-developed by Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba.
>
> ACPI establishes industry-standard interfaces for OS-directed configuration and power management on laptops, desktops, and servers.
>
> ACPI evolves the existing collection of power management BIOS code, Advanced Power Management (APM) application programming interfaces (APIs, PNPBIOS APIs, Multiprocessor Specification (MPS) tables and so on into a well-defined power management and configuration interface specification."[1]

ACPI 1.0 was published in 1996. 2.0 added 64-bit support in 2000. ACPI 3.0 is expected in summer 2004.

2 Linux ACPI Deployment

Linux supports ACPI on three architectures: `ia64`, `i386`, and `x86_64`.

2.1 `ia64` Linux/ACPI support

Most `ia64` platforms require ACPI support, as they do not have the legacy configuration methods seen on `i386`. All the Linux distributions that support `ia64` include ACPI support, whether they're based on Linux-2.4 or Linux-2.6.

2.2 `i386` Linux/ACPI support

Not all Linux-2.4 distributions enabled ACPI by default on `i386`. Often they used just enough table parsing to enable Hyper-Threading (HT), ala `acpi=ht` below, and relied on MPS and PIRQ routers to configure the

[1] `http://www.acpi.info`

```
setup_arch()
  dmi_scan_machine()
    Scan DMI blacklist
    BIOS Date vs Jan 1, 2001
  acpi_boot_init()
    acpi_table_init()
      locate and checksum all ACPI tables
      print table headers to console
    acpi_blacklisted()
      ACPI table headers vs. blacklist
    parse(BOOT) /* Simple Boot Flags */
    parse(FADT) /* PM timer address */
    parse(MADT) /* LAPIC, IOAPIC */
    parse(HPET) /* HiPrecision Timer */
    parse(MCFG) /* PCI Express base */
```

Figure 1: Early ACPI init on `i386`

machine. Some included ACPI support by default, but required the user to add `acpi=on` to the cmdline to enable it.

So far, the major Linux 2.6 distributions all support ACPI enabled by default on `i386`.

Several methods are used to make it more practical to deploy ACPI onto `i386` installed base. Figure 1 shows the early ACPI startup on the `i386` and where these methods hook in.

1. Most modern system BIOS support DMI, which exports the date of the BIOS. Linux DMI scan in `i386` disables ACPI on platforms with a BIOS older than January 1, 2001. There is nothing magic about this date, except it allowed developers to focus on recent platforms without getting distracted debugging issues on very old platforms that:

   (a) had been running Linux w/o ACPI support for years.

   (b) had virtually no chance of a BIOS update from the OEM.

   Boot parameter `acpi=force` is available to enable ACPI on platforms older than the cutoff date.

2. DMI also exports the hardware manufacturer, baseboard name, BIOS version, etc. that you can observe with `dmidecode`.[2] `dmi_scan.c` has a general purpose blacklist that keys off this information, and invokes various platform-specific workarounds. `acpi=off` is the most severe—disabling all ACPI support, even the simple table parsing needed to enable Hyper-Threading (HT). `acpi=ht` does the same, excepts parses enough tables to enable HT. `pci=noacpi` disables ACPI for PCI enumeration and interrupt configuration. And `acpi=noirq` disables ACPI just for interrupt configuration.

3. The ACPI tables also contain header information, which you see near the top of the kernel messages. ACPI maintains a blacklist based on the table headers. But this blacklist is somewhat primitive. When an entry matches the system, it either prints warnings or invokes `acpi=off`.

All three of these methods share the problem that if they are successful, they tend to hide root-cause issues in Linux that should be fixed. For this reason, adding to the blacklists is discouraged in the upstream kernel. Their main value is to allow Linux distributors to quickly react to deployment issues when they need to support deviant platforms.

2.3 `x86_64` Linux/ACPI support

All `x86_64` platforms I've seen include ACPI support. The major `x86_64` Linux distributions, whether Linux-2.4 or Linux-2.6 based, all support ACPI.

[2] `http://www.nongnu.org/dmidecode`

3 Implementation Overview

The ACPI specification describes platform registers, ACPI tables, and operation of the ACPI BIOS. Figure 2 shows these ACPI components logically as a layer above the platform specific hardware and firmware.

The ACPI kernel support centers around the ACPICA (ACPI Component Architecture[3]) core. ACPICA includes the AML[4] interpreter that implements ACPI's hardware abstraction. ACPICA also implements other OS-agnostic parts of the ACPI specification. The ACPICA code does not implement any policy, that is the realm of the Linux-specific code. A single file, `osl.c`, glues ACPICA to the Linux-specific functions it requires.

The box in Figure 2 labeled "Linux/ACPI" represents the Linux-specific ACPI code, including boot-time configuration.

Optional "ACPI drivers," such as Button, Battery, Processor, etc. are (optionally loadable) modules that implement policy related to those specific features and devices.

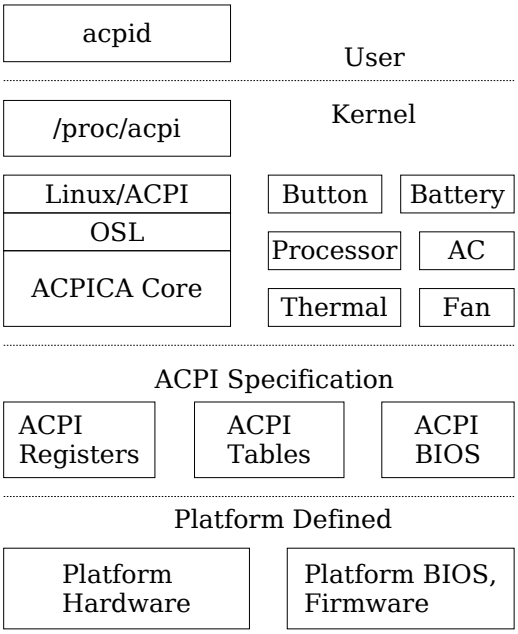

Figure 2: Implementation Architecture

3.1 Events

ACPI registers for a "System Control Interrupt" (SCI) and all ACPI events come through that interrupt.

The kernel interrupt handler de-multiplexes the possible events using ACPI constructs. In some cases, it then delivers events to a user-space application such as `acpid` via `/proc/acpi/events`.

[3] `http://www.intel.com/technology/iapc/acpi`

[4] AML, ACPI Machine Language.

4 ACPI Configuration

Interrupt configuration on `i386` dominated the ACPI bug fixing activity over the last year.

The algorithm to configure interrupts on an `i386` system with an IOAPIC is shown in Figure 3. ACPI mandates that all PIC mode IRQs be identity mapped to IOAPIC pins. Exceptions are specified in MADT [5] interrupt source override entries.

Over-rides are often used, for example, to specify that the 8254 timer on IRQ0 in PIC mode does not use pin0 on the IOAPIC, but uses pin2. Over-rides also often move the ACPI SCI to a different pin in IOAPIC mode than it had in PIC mode, or change its polarity or trigger from the default.

[5] MADT, Multiple APIC Description Table.

```
setup_arch()
 acpi_boot_init()
  parse(MADT);
   parse(LAPIC); /* processors */
   parse(IOAPIC)
    parse(INT_SRC_OVERRIDE);
    add_identity_legacy_mappings();
    /* mp_irqs[] initialized */

init()
 smp_boot_cpus()
  setup_IO_APIC()
   enable_IO_APIC();
   setup_IO_APIC_irqs(); /* mp_irqs[] */
 do_initcalls()
  acpi_init()
"ACPI: Subsystem revision 20040326"
   acpi_initialize_subsystem();
   /* AML interpreter */
   acpi_load_tables(); /* DSDT */
   acpi_enable_subsystem();
   /* HW into ACPI mode */
"ACPI: Interpreter enabled"
   acpi_bus_init_irq();
    AML(_PIC, PIC | IOAPIC | IOSAPIC);

  acpi_pci_link_init()
   for(every PCI Link in DSDT)
    acpi_pci_link_add(Link)
     AML(_PRS, Link);
     AML(_CRS, Link);
"... Link [LNKA] (IRQs 9 10 *11)"

  pci_acpi_init()
"PCI: Using ACPI for IRQ routing"
  acpi_irq_penalty_init();
  for (PCI devices)
   acpi_pci_irq_enable(device)
    acpi_pci_irq_lookup()
     find _PRT entry
      if (Link) {
       acpi_pci_link_get_irq()
        acpi_pci_link_allocate()
         examine possible & current IRQs
          AML(_SRS, Link)
      } else {
        use hard-coded IRQ in _PRT entry
      }
    acpi_register_gsi()
     mp_register_gsi()
      io_apic_set_pci_routing()
"PCI: PCI interrupt 00:06.0[A] ->
  GSI 26 (level, low) -> IRQ 26"
```

Figure 3: Interrupt Initialization

So after identifying that the system will be in IOAPIC mode, the 1st step is to record all the Interrupt Source Overrides in `mp_irqs[]`. The second step is to add the legacy identity mappings where pins and IRQs have not been consumed by the over-rides.

Step three is to digest `mp_irqs[]` in `setup_IO_APIC_irqs()`, just like it would be if the system were running in legacy MPS mode.

But that is just the start of interrupt configuration in ACPI mode. The system still needs to enable the mappings for PCI devices, which are stored in the DSDT[6] _PRT[7] entries. Further, the _PRT can contain both static entries, analogous to MPS table entries, or it can contain dynamic _PRT entries that use PCI Interrupt Link Devices.

So Linux enables the AML interpreter and informs the ACPI BIOS that it plans to run the system in IOAPIC mode.

Next the PCI Interrupt Link Devices are parsed. These “links” are abstract versions of what used to be called PIRQ-routers, though they are more general. `acpi_pci_link_init()` searches the DSDT for Link Devices and queries each about the IRQs it can be set to (_PRS)[8] and the IRQ that it is already set to (_CRS)[9]

A penalty table is used to help decide how to program the PCI Interrupt Link Devices. Weights are statically compiled into the table to avoid programming the links to well known legacy IRQs. `acpi_irq_penalty_init()` updates the table to add penalties to the IRQs where the Links have possible set-

[6] DSDT, Differentiated Services Description Table, written in AML

[7] _PRT, PCI Routing Table

[8] PRS, Possible Resource Settings.

[9] CRS, Current Resource Settings.

tings. The idea is to minimize IRQ sharing, while not conflicting with legacy IRQ use. While it works reasonably well in practice, this heuristic is inherently flawed because it assumes the legacy IRQs rather than asking the DSDT what legacy IRQs are actually in use.[10]

The PCI sub-system calls `acpi_pci_irq_enable()` for every device. ACPI looks up the device in the _PRT by device-id and if it a simple static entry, programs the IOAPIC. If it is a dynamic entry, `acpi_pci_link_allocate()` chooses an IRQ for the link and programs the link via AML (_SRS).[11] Then the associated IOAPIC entry is programmed.

Later, the drivers initialize and call `request_irq(IRQ)` with the IRQ the PCI sub-system told it to request.

One issue we have with this scheme is that it can't automatically recover when the heuristic balancing act fails. For example when the parallel port grabs IRQ7 and a PCI Interrupt Links gets programmed to the same IRQ, then `request_irq(IRQ)` correctly fails to put ISA and PCI interrupts on the same pin. But the system doesn't realize that one of the contenders could actually be re-programmed to a different IRQ.

The fix for this issue will be to delete the heuristic weights from the IRQ penalty table. Instead the kernel should scan the DSDT to enumerate exactly what legacy devices reserve exactly what IRQs.[12]

[10] In PIC mode, the default is to keep the BIOS provided current IRQ setting, unless cmdline `acpi_irq_balance` is used. Balancing is always enabled in IOAPIC mode.

[11] SRS, Set Resource Setting

[12] bugzilla 2733

4.1 Issues With PCI Interrupt Link Devices

Most of the issues have been with PCI Interrupt Link Devices, an ACPI mechanism primarily used to replace the chip-set-specific Legacy PIRQ code.

- The status (_STA) returned by a PCI Interrupt Link Device does not matter. Some systems mark the ones we should use as enabled, some do not.
- The status set by Linux on a link is important on some chip sets. If we do not explicitly disable some unused links, they result in tying together IRQs and can cause spurious interrupts.
- The current setting returned by a link (_CRS) can not always be trusted. Some systems return invalid settings always. Linux must assume that when it sets a link, the setting was successful.
- Some systems return a current setting that is outside the list of possible settings. Per above, this must be ignored and a new setting selected from the possible-list.

4.2 Issues With ACPI SCI Configuration

Another area that was ironed out this year was the ACPI SCI (System Control Interrupt). Originally, the SCI was always configured as level/low, but SCI failures didn't stop until we implemented the algorithm in Figure 4. During debugging, the kernel gained the cmdline option that applies to either PIC or IOAPIC mode: `acpi_sci={level,edge,high,low}` but production systems seem to be working properly and this has seen use recently only to work around prototype BIOS bugs.

```
if (PIC mode) {
  set ELCR to level trigger();
} else { /* IOAPIC mode */
  if (Interrupt Source Override) {
    Use IRQ specified in override
    if(trigger edge or level)
      use edge or level
    else (compatible trigger)
      use level

    if (polarity high or low)
      use high or low
    else
      use low
  } else { /* no Override */
    use level-trigger
    use low-polarity
  }
}
```

Figure 4: SCI configuration algorithm

4.3 Unresolved: Local APIC Timer Issue

The most troublesome configuration issue today is that many systems with no IO-APIC will hang during boot unless their LOCAL-APIC has been disabled, eg. by booting `nolapic`. While this issue has gone away on several systems with BIOS upgrades, entire product lines from high-volume OEMS appear to be subject to this failure. The current workaround to disable the LAPIC timer for the duration of the SMI-CMD update that enables ACPI mode. [13]

4.4 Wanted: Generic Linux Driver Manager

The ACPI DSDT enumerates motherboard devices via PNP identifiers. This method is used to load the ACPI specific devices today, eg. battery, button, fan, thermal etc. as well as `8550_acpi`. PCI devices are enumerated via PCI-ids from PCI config space. Legacy devices probe out using hard-coded address values.

But a device driver should not have to know or care how it is enumerated by its parent bus. An 8250 driver should worry about the 8250 and not if it is being discovered by legacy means, ACPI enumeration, or PCI.

[13] `http://bugzilla.kernel.org 1269`

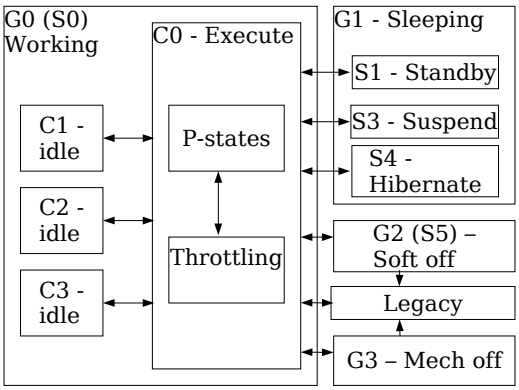

Figure 5: ACPI Global, CPU, and Sleep states.

One fix would be to be to abstract the PCI-ids, PNP-ids, and perhaps even some hard-coded values into a generic device manager directory that maps them to device drivers.

This would simply add a veneer to the PCI device configuration, simplifying a very small number of drivers that can be configured by PCI or ACPI. However, it would also fix the real issue that the configuration information in the ACPI DSDT for most motherboard devices is currently not parsed and not communicated to any Linux drivers.

The Device driver manager would also be able to tell the power management sub-system which methods are used to power-manage the device. Eg. PCI or ACPI.

5 ACPI Power Management

The Global System States defined by ACPI are illustrated in Figure 5. G0 is the working state, G1 is sleeping, G2 is soft-off and G3 is mechanical off. The “Legacy” state illustrates where the system is not in ACPI mode.

5.1 P-states

In the context of G0 – Global Working State, and C0 – CPU Executing State, P-states (Performance states) are available to reduce power of the running processor. P-states simultaneously modulate both the MHz and the voltage. As power varies by voltage squared, P-states are extremely effective at saving power.

While P-states are extremely important, the `cpufreq` sub-system handles P-states on a number of different platforms, and the topic is best addressed in that larger context.

5.2 Throttling

In the context of the G0-Working, C0-Executing state, Throttling states are defined to modulate the frequency of the running processor.

Power varies (almost) directly with MHz, so when the MHz is cut if half, so is the power. Unfortunately, so is the performance.

Linux currently uses Throttling only in response to thermal events where the processor is too hot. However, in the future, Linux could add throttling when the processor is already in the lowest P-state to save additional power.

Note that most processors also include a backup Thermal Monitor throttling mechanism in hardware, set with higher temperature thresholds than ACPI throttling. Most processors also have in hardware an thermal emergency shutdown mechanism.

5.3 C-states

In the context of G0 Working system state, C-state (CPU-state) C0 is used to refer to the executing state. Higher number C-states are entered to save successively more power when the processor is idle. No instructions are executed when in C1, C2, or C3.

ACPI replaces the default idle loop so it can enter C1, C2 or C3. The deeper the C-state, the more power savings, but the higher the latency to enter/exit the C-state. You can observe the C-states supported by the system and the success at using them in `/proc/acpi/processor/CPU0/power`

C1 is included in every processor and has negligible latency. C1 is implemented with the HALT or MONITOR/MWAIT instructions. Any interrupt will automatically wake the processor from C1.

C2 has higher latency (though always under 100 usec) and higher power savings than C1. It is entered through writes to ACPI registers and exits automatically with any interrupt.

C3 has higher latency (though always under 1000 usec) and higher power savings than C2. It is entered through writes to ACPI registers and exits automatically with any interrupt or bus master activity. The processor does not snoop its cache when in C3, which is why bus-master (DMA) activity will wake it up. Linux sees several implementation issues with C3 today:

1. C3 is enabled even if the latency is up to 1000 usec. This compares with the Linux 2.6 clock tick rate of 1000Hz = 1ms = 1000usec. So when a clock tick causes C3 to exit, it may take all the way to the next clock tick to execute the next kernel instruction. So the benefit of C3 is lost because the system effectively pays C3 latency and gets negligible C3 residency to save power.

2. Some devices do not tolerate the DMA latency introduced by C3. Their device buffers underrun or overflow. This is cur-

rently an issue with the ipw2100 WLAN NIC.

3. Some platforms can lie about C3 latency and transparently put the system into a higher latency C4 when we ask for C3—particularly when running on batteries.

4. Many processors halt their local APIC timer (a.k.a. TSC – Timer Stamp Counter) when in C3. You can observe this by watching LOC fall behind IRQ0 in /proc/interrupts.

5. USB makes it virtually impossible to enter C3 because of constant bus master activity. The workaround at the moment is to unplug your USB devices when idle. Longer term, it will take enhancements to the USB sub-system to address this issue. Ie. USB software needs to recognize when devices are present but idle, and reduce the frequency of bus master activity.

Linux decides which C-state to enter on idle based on a promotion/demotion algorithm. The current algorithm measures the residency in the current C-state. If it meets a threshold the processor is promoted to the deeper C-state on re-entrance into idle. If it was too short, then the processor is demoted to a lower-numbered C-state.

Unfortunately, the demotion rules are overly simplistic, as Linux tracks only its previous success at being idle, and doesn't yet account for the load on the system.

Support for deeper C-states via the _CST method is currently in prototype. Hopefully this method will also give the OS more accurate data than the FADT about the latency associated with C3. If it does not, then we may need to consider discarding the table-provided latencies and measuring the actual latency at boot time.

5.4 Sleep States

ACPI names sleeps states S0 – S5. S0 is the non-sleep state, synonymous with G0. S1 is standby, it halts the processor and turns off the display. Of course turning off the display on an idle system saves the same amount of power without taking the system off line, so S1 isn't worth much. S2 is deprecated. S3 is suspend to RAM. S4 is hibernate to disk. S5 is soft-power off, AKA G2.

Sleep states are unreliable enough on Linux today that they're best considered "experimental." Suspend/Resume suffers from (at least) two systematic problems:

- `_init()` and `_initdata()` on items that may be referenced after boot, say, during resume, is a bad idea.

- PCI configuration space is not uniformly saved and restored either for devices or for PCI bridges. This can be observed by using `lspci` before and after a suspend/resume cycle. Sometimes `setpci` can be used to repair this damage from user-space.

5.5 Device States

Not shown on the diagram, ACPI defines power saving states for devices: D0 – D3. D0 is on, D3 is off, D1 and D2 are intermediate. Higher device states have

1. more power savings,

2. less device context saved by hardware,

3. more device driver state restoring,

4. higher restore latency.

ACPI defines semantics for each device state in each device class. In practice, D1 and D2 are often optional - as many devices support only on and off either because they are low-latency, or because they are simple.

Linux-2.6 includes an updated device driver model to accommodate power management.[14] This model is highly compatible with PCI and ACPI. However, this vision is not yet fully realized. To do so, Linux needs a global power policy manager.

5.6 Wanted: Generic Linux Run-time Power Policy Manager

PCI device drivers today call `pci_set_power_state()` to enter D-states. This uses the power management capabilities in the PCI power management specification.

The ACPI DSDT supplies methods for ACPI enumerated devices to access ACPI D-states. However, no driver calls into ACPI to enter D-states today.[15]

Drivers shouldn't have to care if they are power managed by PCI or by ACPI. Drivers should be able to up-call to a generic run-time power policy manager. That manager should know about calling the PCI layer or the ACPI layer as appropriate.

The power manager should also put those requests in the context of user-specified power policy. Eg. Does the user want maximum performance, or maximum battery life? Currently there is no method to specify the detailed policy, and the kernel wouldn't know how to handle it anyway.

In a related point, it appears that devices currently only suspend upon system suspend. This is probably not the path to industry leading battery life.

Device drivers should recognize when their device has gone idle. They should invoke a suspend up-call to a power manager layer which will decide if it really is a good idea to grant that request now, and if so, how. In this case by calling the PCI or ACPI layer as appropriate.

6 ACPI as seen by bugzilla

Over the last year the ACPI developers have made heavy use of bugzilla[16] to help prioritize and track 460 bugs. 300 bugs are closed or resolved, 160 are open. [17]

We cc: `acpi-bugzilla@lists.sourceforge.net` on these bugs, and we encourage the community to add that alias to ACPI-specific bugs in other bugzillas so that the team can help out wherever the problems are found.

We haven't really used the bugzilla priority field. Instead we've split the bugs into categories and have addressed the configuration issues first. This explains why most of the interrupt bugs are resolved, and most of the suspend/resume bugs are unresolved.

We've seen an incoming bug rate of 10-bugs/week for many months, but the new reports favor the power management features over configuration, so we're hopeful that the torrent of configuration issues is behind us.

[14] Patrick Mochel, Linux Kernel Power Management, OLS 2003.

[15] Actually, the ACPI hot-plug driver invokes D-states, but that is the only exception.

[16] `http://bugzilla.kernel.org/`

[17] The resolved state indicates that a patch is available for testing, but that it is not yet checked into the kernel.org kernel.

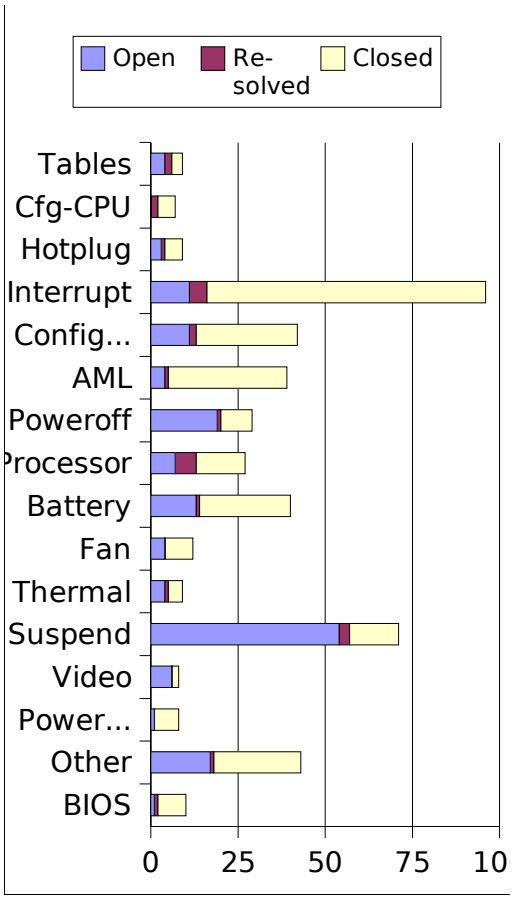

Figure 6: ACPI bug profile

7 Future Work

7.1 Linux 2.4

Going forward, I expect to back-port only critical configuration related fixes to Linux-2.4. For the latest power management code, users need to migrate to Linux-2.6.

7.2 Linux 2.6

Linux-2.6 is a "stable" release, so it is not appropriate to integrate significant new features. However, the power management side of ACPI is widely used in 2.6 and there will be plenty of bug-fixes necessary. The most visible will probably be anything that makes Suspend/Resume work on more platforms.

7.3 Linux 2.7

These feature gaps will not be addressed in Linux 2.6, and so are candidates for Linux 2.7:

- Device enumeration is not abstracted in a generic device driver manager that can shield drivers from knowing if they're enumerated by ACPI, PCI, or other.
- Motherboard devices enumerated by ACPI in the DSDT are ignored, and probed instead via legacy methods. This can lead to resource conflicts.
- Device power states are not abstracted in a generic device power manager that can shield drivers from knowing whether to call ACPI or PCI to handle D-states.
- There is no power policy manager to translate the user-requested power policy into kernel policy.
- No devices invoke ACPI methods to enter D-states.
- Devices do not detect that they are idle and request of a power manager whether they should enter power saving device states.
- There is no MP/SMT coordination of P-states. Today, P-states are disabled on SMP systems. Coordination needs to account for multiple threads and multiple cores per package.
- Coordinate P-states and T-states. Throttling should be used only after the system is put in the lowest P-state.
- Idle states above C1 are disabled on SMP.
- Enable Suspend in PAE mode. [18]

[18] PAE, Physical Address Extended—MMU mode to handle > 4GB RAM—optional on `i386`, always used on `x86_64`.

- Enable Suspend on SMP.
- Tick timer modulation for idle power savings.
- Video control extensions. Video is a large power consumer. The ACPI spec Video extensions are currently in prototype.
- Docking Station support is completely absent from Linux.
- ACPI 3.0 features. TBD after the specification is published.

7.4 ACPI 3.0

Although ACPI 3.0 has not yet been published, two ACPI 3.0 tidbits are already in Linux.

- PCI Express table scanning. This is the basic PCI Express support, there will be more coming. Those in the PCI SIG can read all about it in the PCI Express Firmware Specification.
- Several clarifications to the ACPI 2.0b spec resulted directly from open source development, [19] and the text of ACPI 3.0 has been updated accordingly. For example, some subtleties of SCI interrupt configuration and device enumeration.

When the ACPI 3.0 specification is published there will instantly be a multiple additions to the ACPI/Linux feature to-do list.

7.5 Tougher Issues

- Battery Life on Linux is not yet competitive. This single metric is the sum of all the power savings features in the platform, and if any of them are not working properly, it comes out on this bottom line.
- Laptop Hot Keys are used to control things such as video brightness, etc. ACPI does not specify Hot Keys. But when they work in APM mode and don't work in ACPI mode, ACPI gets blamed. There are 4 ways to implement hot keys:
  1. SMI[20] handler, the BIOS handles interrupts from the keys, and controls the device directly. This acts like "hardware" control as the OS doesn't know it is happening. But on many systems this SMI method is disabled as soon as the system transitions into ACPI mode. Thus the complaint "the button works in APM mode, but doesn't work in ACPI mode."

     But ACPI doesn't specify how hot keys work, so in ACPI mode one of the other methods listed here needs to handle the keys.
  2. Keyboard Extension driver, such as `i8k`. Here the keys return scan codes like any other keys on the keyboard, and the keyboard driver needs to understand those scan code. This is independent of ACPI, and generally OEM specific.
  3. OEM-specific ACPI hot key driver. Some OEMs enumerate the hot keys as OEM-specific devices in the ACPI tables. While the device is described in AML, such devices are not described in the ACPI spec so we can't build generic ACPI support for them. The OEM must supply the appropriate hot-key driver since only they know how it is supposed to work.
  4. Platform-specific "ACPI" driver. Today Linux includes Toshiba and

[19] FreeBSD deserves kudos in addition to Linux

[20] SMI, System Management Interrupt; invisible to the OS, handled by the BIOS, generally considered evil.

Asus platform specific extension drivers to ACPI. They do not use portable ACPI compliant methods to recognize and talk to the hot keys, but generally use the methods above.

The correct solution to the the Hot Key issue on Linux will require direct support from the OEMs, either by supplying documentation, or code to the community.

8 Summary

This past year has seen great strides in the configuration aspects of ACPI. Multiple Linux distributors now enable ACPI on multiple architectures.

This sets the foundation for the next era of ACPI on Linux where we can evolve the more advanced ACPI features to meet the expectations of the community.

9 Resources

The ACPI specification is published at `http://www.acpi.info`.

The home page for the Linux ACPI development community is here: `http://acpi.sourceforge.net/` It contains numerous useful pointers, including one to the `acpi-devel` mailing list.

The latest ACPI code can be found against various recent releases in the BitKeeper repositories: `http://linux-acpi.bkbits.net/`

Plain patches are available on `kernel.org`.[21] Note that Andrew Morton currently includes the latest ACPI test tree in the `-mm` patch, so you can test the latest ACPI code combined with other recent updates there.[22]

10 Acknowledgments

Many thanks to the following people whose direct contributions have significantly improved the quality of the ACPI code in the last year: Jesse Barnes, John Belmonte, Dominik Brodowski, Bruno Ducrot, Bjorn Helgaas, Nitin, Kamble, Andi Kleen, Karol Kozimor, Pavel Machek, Andrew Morton, Jun Nakajima, Venkatesh Pallipadi, Nate Lawson, David Shaohua Li, Suresh Siddha, Jes Sorensen, Andrew de Quincey, Arjan van de Ven, Matt Wilcox, and Luming Yu. Thanks also to all the bug submitters, and the enthusiasts on `acpi-devel`.

Special thanks to Intel's Mobile Platforms Group, which created ACPICA, particularly Bob Moore and Andy Grover.

[21] `http://ftp.kernel.org/pub/linux/kernel/people/lenb/acpi/patches/`

[22] `http://ftp.kernel.org/pub/linux/kernel/people/akpm/patches/`