Cool Hand Linux® – Handheld Thermal Extensions

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

Harinarayanan Seshadri
Intel Ultra-Mobile Group
harinarayanan.seshadri@intel.com

Abstract

Linux has traditionally been centered around processor performance and power consumption. Thermal management has been a secondary concern—occasionally used for fan speed control, sometimes for processor throttling, and once in a rare while for an emergency thermal shutdown.

Handheld devices change the rules. Skin temperature is a dominant metric, the processor may be a minority player in heat generation, and there are no fans.

This paper describes extensions to Linux thermal management to meet the challenges of handheld devices.

1 Handhelds Thermal Challenges

The new generation handheld computing devices are exploding with new usage models like navigation, infotainment, health, and UMPC. "Internet on the Go" is a common feature set the handheld devices must provide for all these new usage models. This requires handheld devices to be high-performance.

High-performance handhelds have magnified power and thermal challenges as compared to notebooks.

- Notebooks today are infrequently designed to sit in your lap. Most of them are designed to sit on a table, and many actually require free air flow from beneath the case. Users demand that handheld computers be cool enough that their hand does not sweat. Thus, there are strict skin temperature limits on handhelds.
- Notebooks typically have fans. Handhelds typically do not have fans.
- Handheld form factors are physically smaller than a typical notebook, and thus the thermal dissipation within the platform is limited.
- The CPU is the dominant heat generator on most notebooks. But for handhelds, the CPU may be a minor contributor as compared to other devices.

2 ACPI Thermal Capabilities

Linux notebooks today use a combination of ACPI and native-device thermal control.

The ACPI specification [ACPI] mandates that if a notebook has both active and passive cooling capability, then the platform must expose them to the OS via ACPI. The reasoning behind this is that the OS should be involved in the cooling policy decision when deciding between active versus passive cooling.

But a large number of notebooks implement thermal control "behind the scenes" and don't inform or involve ACPI or the OS at all. Generally this means that they control the fan(s) via chipset or embedded controller; but in some cases, they go further, and also implement thermal throttling in violation of the ACPI spec.

2.1 Linux will not use the ACPI 3.0 Thermal Extensions

ACPI 3.0 added a number of thermal extensions—collectively called the "3.0 Thermal Model." These extensions include the ability to relate the relative contributions of multiple devices to multiple thermal zones. These relative contributions are measured at system design-time, and encoded in an ACPI thermal relationship table for use by the OS in balancing the multiple contributors to thermal load. This is a sophisticated solution requiring thorough measurements by the system designer, as well as knowledge on the part of the OS about relative performance tradeoffs.

The handheld effort described in this paper does not need the ACPI 3.0 thermal extensions. Indeed, no Linux

notebook has yet materialized that needs those extensions.

So when the BIOS AML uses `_OSI` and asks Linux if it supports the "3.0 Thermal Model," Linux will continue to answer "no," for the immediate future.

2.2 How the ACPI 2.0 Thermal Model works

ACPI defines a concept of thermal zones. A thermal zone is effectively a thermometer with associated trip points and devices.

A single CRT trip point provokes a critical system shutdown. A single HOT trip point provokes a system suspend-to-disk. A single PSV trip point activates the OS's passive cooling algorithm. One or more ACx trip points control fans for active cooling. Each active trip point may be associated with one or multiple fans, or with multiple speeds of the same fan.

When a PSV trip point fires, the Linux `processor_thermal` driver receives the event and immediately requests the Linux cpufreq subsystem to enter the deepest available processor performance state (P-state). As P-states reduce voltage along with frequency, they are more power-efficient than simple clock throttling (T-states), which lower frequency only.

The processor thermal throttling algorithm then periodically polls the thermal zone for temperature, and throttles the clock accordingly. When the system has cooled and the algorithm has run its course, the processor is un-throttled, and cpufreq is again allowed to control P-states.

2.3 Why is the ACPI 2.0 Thermal Model not sufficient?

It can't hurt to implement ACPI's CRT trip point for critical system shutdown—but that isn't really the focus here.

The HOT trip point doesn't really make sense, since on a handheld, shutdown and hibernate to disk (if one even exists) are likely to be synonymous.

Active trip points are of no use on systems which have no fans.

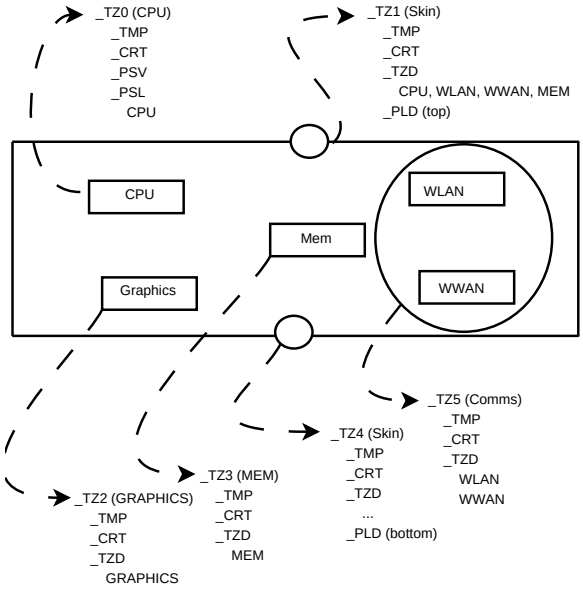

Figure 1: Example Sensor-to-Thermal-Zone Mapping

That leaves the single PSV trip point. ACPI 2.0 can associate (only) a processor throttling device with a trip point. Yes, handhelds have processors, but the processor isn't expected to always be the dominant contributor to thermal footprint on handhelds like it often is on notebooks.

ACPI 2.0 includes the `_TZD` method to associate devices with thermal zones. However, ACPI doesn't say anything about how to throttle non-processor devices—so that must be handled by native device drivers.

2.4 So why use ACPI at all?

Inexpensive thermal sensors do not know how to generate events. They are effectively just thermometers that need to be polled. However, using the CPU to poll the sensors would be keeping a relatively power-hungry component busy on a relatively trivial task. The solution is to poll the sensors from a low-power embedded controller (EC).

The EC is always running. It polls all the sensors and is responsible for interrupting the main processor with events. ACPI defines a standard EC, and so it is easy to re-use that implementation. Of course, it would be an equally valid solution to use a native device driver to talk to an on-board microcontroller that handles the low-level polling.

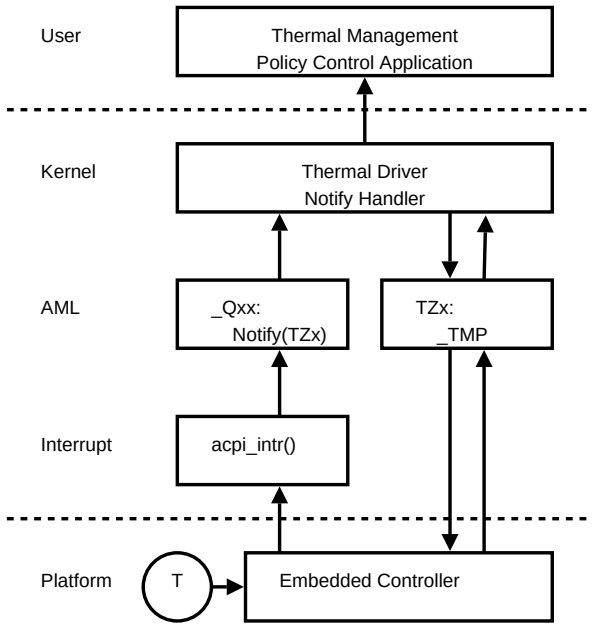

Figure 2: Thermal Event Delivery via ACPI

2.5 Mapping Platform Sensors to ACPI Thermal Zones

Figure 1 shows an example of mapping platform sensors to the ACPI Thermal zones. Four different scenarios are illustrated in the figure:

- Sensors that are built into the CPU.
- Sensors that are associated with a single non-processor device, such as DRAM or graphics.
- Sensors that are associated with cluster of components, for example, Wireless LAN (WLAN) and Wireless WAN (WWAN).
- Skin sensors that indicate overall platform temperature.

2.6 How to use ACPI for Handheld Thermal Events

For the CPU, Linux can continue to handle a single ACPI passive trip point with an in-kernel processor thermal throttling algorithm.

For critical thermal events, Linux can continue to handle a single ACPI critical trip point with a system shutdown.

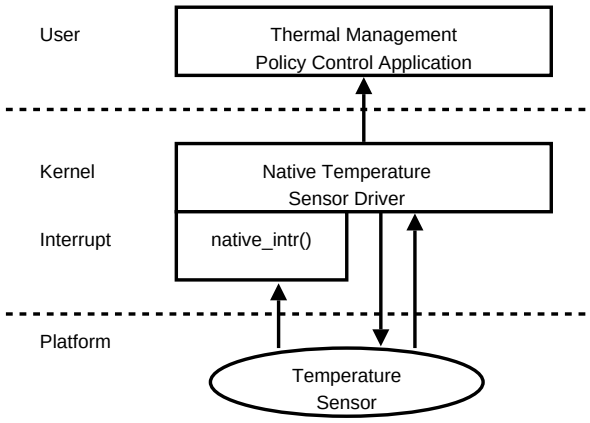

Figure 3: Thermal Event Delivery via native driver

However, for the non-processor thermal zones, a single passive trip point is insufficient. For those we will use ACPI's concept of "temperature change events."

When the EC decides that the temperature has changed by a meaningful amount—either up or down—it sends a temperature change event to the thermal zone object.

If the thermal zone is associated with a processor, then the kernel can invoke its traditional processor thermal throttling algorithm.

As shown in Figure 2, for non-processor thermal zones, the thermal driver will query the temperature of the zone, and send a netlink message to user-space identifying the zone and the current temperature.

Figure 3 shows the same event delivered by a native platform, sensor-specific sensor driver.

3 Proposed Handheld Thermal Solution

Multiple works ([LORCH], [VAHDAT]) focus on low-power OS requirements and low-power platform design. While low-power optimizations have a positive impact on thermals, this only addresses thermal issues at a component level. To address the platform-level thermal issues, we need to look at the thermal problem in a more complete platform perspective. This requires support from OS, user applications, etc.

3.1 Design Philosophy

- Thermal monitoring will be done using inexpensive thermal sensors—polled by a low-power EC.

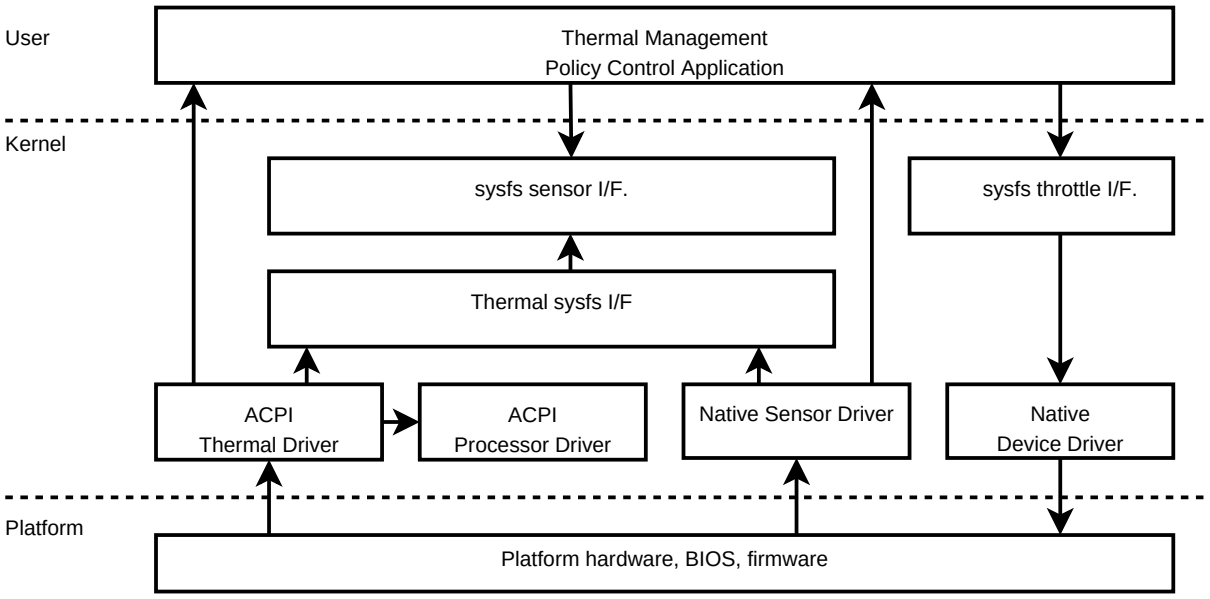

Figure 4: Thermal Zone Driver Stack Architecture

- Thermal management policy decisions will be made from user space, as the user has a comprehensive view of the platform.

- The kernel provides only the mechanism to deliver thermal events to user space, and the mechanism for user space to communicate its throttling decisions to native device drivers.

Figure 4 shows the thermal control software stack. The thermal management policy control application sits on top. It receives netlink messages from the kernel thermal zone driver. It then implements device-specific thermal throttling via sysfs. Native device drivers supply the throttling controls in sysfs and implement device-specific throttling functions.

3.2 Thermal Zone module

The thermal zone module has two components—a thermal zone sysfs driver and thermal zone sensor driver. The thermal zone sysfs driver is platform-independent, and handles all the sysfs interaction. The thermal zone sensor driver is platform-dependent. It works closely with the platform BIOS and sensor driver, and has knowledge of sensor information in the platform.

3.2.1 Thermal sysfs driver

The thermal sysfs driver exports two interfaces (`thermal_control_register()` and `thermal_control_deregister()`) to component drivers, which the component drivers can call to register their control capability to the thermal zone sysfs driver.

The thermal sysfs driver also exports two interfaces—`thermal_sensor_register()` and `thermal_sensor_deregister()`—to the platform-specific sensor drivers, where the sensor drivers can use this interface to register their sensor capability.

This driver is responsible for all thermal sysfs entries. It interacts with all the platform specific thermal sensor drivers and component drivers to populate the sysfs entries.

The thermal zone driver also provides a notification-of-temperature service to a component driver. The thermal zone sensor driver as part of registration exposes its sensing and thermal zone capability.

3.2.2 Thermal Zone sensor driver

The thermal zone sensor driver provides all the platform-specific sensor information to the thermal

sysfs driver. It is platform-specific in that it has prior information about the sensors present in the platform.

The thermal zone driver directly maps the ACPI 2.0 thermal zone definition, as shown in Figure 1. The thermal zone sensor driver also handles the interrupt notification from the sensor trips and delivers it to user space through netlink socket.

3.3 Component Throttle driver

All the component drivers participating in the given thermal zone can register with the thermal driver, each providing the set of thermal ops it can support. The thermal driver will redirect all the control requests to the appropriate component drivers when the user programs the throttling level. Its is up to the component driver to implement the thermal control.

For example, a component driver associated with DRAM would slow down the DRAM clock on throttling requests.

3.4 Thermal Zone Sysfs Property

Table 1 shows the directory structure exposing each thermal zone sysfs property to user space.

The intent is that any combination of ACPI and native thermal zones may exist on a platform, but the generic sysfs interface looks the same for all of them. Thus, the syntax of the files borrows heavily from the Linux `hwmon` sub-system.[1]

Each thermal zone provides its current temperature and an indicator that can be used by user-space to see if the current temperature has changed since the last read.

If a critical trip point is present, its value is indicated here, as well as an alarm indicator showing whether it has fired.

If a passive trip point is present, its value is indicated here, as well as an alarm indicator showing whether it has fired.

There are symbolic links to the device nodes of the devices associated with the thermal zone. Those devices will export their throttling controls under their device nodes.

[1] `Documentation/hwmon/sysfs-interface` defines the names of the sysfs files, except for the passive files, which are new.

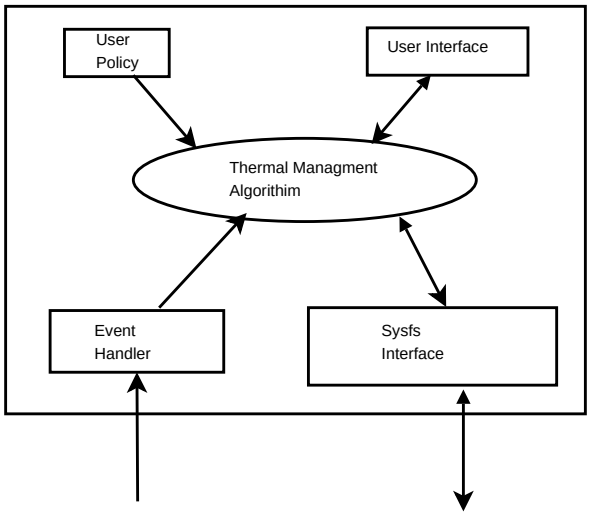

Figure 5: Thermal Policy Control Application

3.5 Throttling Sysfs Properties

Devices that support throttling will have two additional properties associated with the device nodes: `throttling` and `throttling_max`.

A value of 0 means maximum performance, though no throttling. A value of `throttling_max` means maximum power savings in the deepest throttling state available before device state is lost.

3.6 Netlink Socket Kobject event notification

Events will be passed from the kernel to user-space using the Linux netlink facility. Interrupts from the sensor or EC are delivered to user-space through a netlink socket.

3.7 Policy Control Application

Figure 5 shows a thermal policy control application.

The control application interacts with the user, via GUI or configuration files, etc., such that it can understand both the dependencies within the system, and the desired current operating point of the system.

The thermal management module can monitor the platform and component temperature through the sysfs interface, which is a simple wrapper around the sysfs layer. The application receives temperature change

sysfs	ACPI	Description	R/W
temp1_input	_TMP	Current temperature	RO
temp1_alarm		Temperature change occurred	RW
temp1_crit	_CRT	Critical alarm temperature	RO
temp1_crit_alarm		Critical alarm occurred	RW
temp1_passive	_PSV	Passive alarm temperature	RO
temp1_passive_alarm		Passive alarm occurred	RW
<device_name1>		Link to device1 associated with zone	RO
<device_name2>		Link to device2 associated with zone	RO
...		...	RO

Table 1: Thermal Zone sysfs entries

events via netlink, so it can track temperature trends. When it decides to implement throttling, it accesses the appropriate native device's sysfs entries via the sysfs interface.

The thermal throttling algorithms implemented internally to the policy control application are beyond the scope of this paper.

3.8 Possible ACPI extensions

This proposal does make use of the ACPI critical trip point. Depending on the device, the policy manager may decide that either the device or the entire system must be shut down in response to a critical trip point.

This proposal also retains the ACPI 2.0 support for a passive trip point associated with a processor, and in-kernel thermal throttling of the processor device.

However, the main use of ACPI in this proposal is simply as a conduit that associates interesting temperature change events with thermal zones.

What is missing from ACPI is a way for the policy manager to tell the firmware via ACPI what events are interesting.

As a result, the EC must have built-in knowledge about what temperature change events are interesting across the operating range of the device.

However, it would be more flexible if the policy control application could simply dictate what granularity of temperature change events it would like to see surrounding the current temperature.

For example, when the temperature is 20C, the policy application may not care about temperature change events smaller than 5C. But when the temperature is higher, change events of 0.5C may be needed for fine control of the throttling algorithm.

4 Conclusion

On handheld computers it is viable for a platform-specific control application to manage the thermal policy. With the policy moved to user-space, the kernel component of the solution is limited to delivering events and exposing device-specific throttling controls.

This approach should be viable on a broad range of systems, both with and without ACPI support.

References

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

[LORCH] J. Lorch and A.J. Smith. *Software Strategies for Portable Computer Energy Management*, IEEE Personal Communications Magazine, 5(3):60-73, June 1998.

[VAHDAT] Vahdat, A., Lebeck, A., and Ellis, C.S. 2000. *Every joule is precious: the case for revisiting operating system design for energy efficiency*, Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges For the Operating System, Kolding, Denmark, September 17–20, 2000. EW 9. ACM Press, New York, NY, 31–36. `http://doi.acm.org/10.1145/566726.566735`

Proceedings of the Linux Symposium

Volume One

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

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
Dirk Hohndel, *Intel*
Martin Bligh, *Google*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
John Feeney, *Red Hat, Inc.*
Len DiMaggio, *Red Hat, Inc.*
John Poelstra, *Red Hat, Inc.*