Quiet Computing with BSD:

Fan control with sysctl hw.sensors.

Constantine A. Murenin and Raouf Boutaba
University of Waterloo

February 2010

Abstract

We will discuss the topic of fan control and introduce sysctl-based interfacing with the fan-controlling capabilities of microprocessor system hardware monitors on OpenBSD. The discussed prototype implementation reduces the noise and power-consumption characteristics in fans of personal computers, especially of those PCs that are designed from off-the-shelf components. We further argue that our prototype is easier, robuster and more intuitive to use compared to solutions available elsewhere.

1. Introduction and Motivation

Power consumption and heat dissipation are gaining widespread attention in many sectors where personal computers are deployed. The process of transferring the heat away from the system is usually accomplished with the help of some combination of fans. However, fans themselves are known to significantly contribute to the total power consumption of the system, and also pose an additional problem of emitting a persistent background noise, which, in turn, is believed to increase the stress levels of those who are exposed to the noise for prolonged periods of time and decrease the motivation of the workforce [Evans.stress].

Big-name system integrators have often been solving the problem of balancing the noise and thermal characteristics by carefully choosing the fans that are used to cool the system down, together with employing some proprietary fan-controlling logic that automatically adjusts the speed of the fans as needed, in order to achieve the lowest possible noise levels without compromising the thermal zone requirements of the system.

However, smaller integrators, whether individual users or companies assembling personal computers from off-the-shelf components, have to play it safe, and oftentimes have to install excessive fans, even if such fans are not strictly necessary for maintaining reliable operation of the computer, due to uncertainties regarding the power consumption of individual components that are used to assemble the system. In essence, this results in unnecessary noise, whereas the opposite approach of putting fewer than necessary fans may result in overheating problems when the computational capabilities of the system are fully utilised for an extended time period. Fortunately, not everything is lost, and many off-the-shelf motherboards feature an integrated hardware monitoring silicon chip that could also allow the user to control the voltages that are supplied to the fans, thus allowing the user to have more control over the environmental characteristics inside of their PC.

Opponents of fan control may cite various reasons against decreasing fan-speed and increasing the operating temperature of the system. A commonly cited negative factor from such a group is the decrease in the product life span of certain components, especially and most importantly Hard Disc Drives (HDDs). An observation that was perhaps relevant in the past is now often disputed — a recent study published by Google Inc suggests that temperature and activity levels are much less correlated with drive failures than previously reported. [Google.FAST07] As for other components of the system, most of them are rarely considered as critical and irreplaceable as the HDDs, and most often they are specifically designed and advertised to operate in rather extreme temperature conditions anyway. Proponents and practitioners of quiet computing can also easily provide the empirical proof that their systems, if configured for technically-reasonable upper-level temperatures, can run stable and reliable for a number of years to come.

In this paper, we will outline some features and problems with these hardware monitoring chips and with their use in the commonplace hardware as it relates to fan control, and we will discuss interfacing options

that allow the end-user to conveniently communicate and enforce their thermal policies.

2. Related Work

In this section, we will provide an overview of some related works that allow the user to specify thermal characteristics of their system.

2.1. Interfacing from the BIOS

Although popular off-the-shelf motherboards have been physically supporting at least some fan-controlling features for a considerably long time now, it has not been until recently that these boards have been bundled with BIOSes that allowed any kind of interfacing with the fan-controlling characteristics of the hardware monitoring chips.

Although most modern BIOSes that are included with the new motherboards do allow the user to monitor the temperature, fan and voltage characteristics of the board through the hardware monitoring chip, it is still not universally common to find boards that feature adequate level of fan-controlling interfacing from within the BIOS menus: some boards don't have any configurable options at all, whereas others simply have an On / Off switch regarding some brand-name "Quiet Fan" feature from the manufacturer of the motherboard.

As a specific example, let us mention a relatively popular Mini-ITX board that is sold under the Intel brand: Intel D201GLY2. [Intel.D201GLY2] Our sample board has been purchased from newegg.com in December 2007 and originally came with the 0122 BIOS revision dated 2007-08-22. As far as the hardware monitoring and fan controlling goes, the original BIOS only had the "Hardware Monitoring" menu selection in its Advanced System Setup tab, with no options for fan control. However, from the release notes that accompany later BIOSes, we could see that "System Fan Control" option has been introduced in a new "Advanced -> Fan Control Configuration" menu of the 0129 BIOS revision from 2007-10-08. In order to test the new option, we have updated the BIOS of this board to the very latest revision numbered 0149 and dated 2008-12-16. After updating the BIOS and going into the "Advanced -> Fan Control Configuration" menu, we have been rather disappointed to find out that the "System Fan Control" is the only option that has now been implemented, and the only parameters it can have is "Enabled" or "Disabled". For illustration purposes, please refer to *Figure I*. In further sections, we will show that the hardware monitoring chip itself has many more fan-controlling options that may be of specific and reasonable interest to the user.

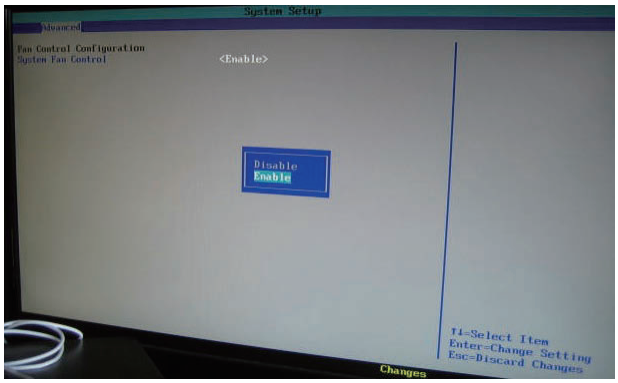

Figure I. A sample screenshot of the options that have been provided in the BIOS revision 0149 dated 2008-12-16 of the Intel D201GLY2 board, which was one of the boards we have used in testing our prototype.

2.2. ACPI

ACPI was introduced with the intention of providing a unified interface for various hardware discovery and power management functions. [Watanabe.ACPI] [Intel.ACPI] However, the reality of modern implementations shows that fan controlling is not one of the functions that is universally provided by ACPI, at least not on the common desktop and server off-the-shelf motherboards.

Laptops, on the other hand, may sometimes feature more useful details regarding environmental characteristics in their ACPI Differentiated System Description Tables. This may include the ACPI Thermal Zones (the concept is not specifically unique to laptops, but in practice is much less common to be present in the desktop boards). Thermal zones may optionally have a number of Active Cooling objects, which define temperature thresholds at which Fan Devices are engaged. Each Fan Device, in this sense, may be a separate physical device, or may represent a logical setting of a varying speed on a single fan (or a set of fans). In practice, however, most of this functionality is still not implemented in the ACPI tables of available hardware; moreover, it is not even clear if such functionality may be found useful for general-purpose off-the-shelf motherboards, where the creator of the ACPI DSDT (in other words, the motherboard manufacturer) may not possibly be aware of the thermal characteristics and active cooling requirements of a custom-build box. For this reason, we would leave further discussion regarding ACPI Thermal Management for a future discourse.

It should be noted, however, that some brand-name laptops and motherboards do have interesting infor-

mation in their ACPI DSDT that may relate to the topic of fan control. Notable examples of such systems include IBM/Lenovo ThinkPad's, ASUSTeK motherboards with the AI Booster / ATK0110 feature and ABIT motherboards with the ABIT uGuru feature. Let us briefly describe the better known features that are provided through these DSDTs.

Out of our interest to environmental monitoring and fan control, ThinkPad's provide multiple temperature sensors (as many as 16 in total), one fan RPM sensor and one fan-speed control setting. The speed control setting could be set to 'automatic', 'disengaged' (meaning, no control is done and fan is run at 100%) and the 'manual' mode. In the manual mode, the setting can be varied between 0 and 7, where values above zero seem to guarantee that the fan actually runs [thinkwiki], meaning that ThinkPads are, essentially, fool-proof, unlike the manual duty cycle mode in chips of custom-build systems from the off-the-shelf components, which are unlikely to make the fans run under less than 40% of the duty cycle (i.e. under 5 V).

ASUSTeK's ACPI ASOC ATK0110 virtual device provides the fan-controlling settings that are similar or identical to those outlined in the BIOS. Currently, there is no open-source implementation that supports the fan-controlling features of the device, so the fan-control interfacing has to be done through the BIOS. In general, although the settings on some of these boards may usually be adequate for many or even most users, they are still rather limited compared to the settings that individual hardware monitoring silicon chips can provide.

However, the task of exploring in great detail the fan controlling interfaces specifically to ACPI we will leave for future work, noting that our current work should provide a solid ground for any further fan controlling enhancements in drivers other than those that we will specifically mention. As we discussed above, currently ACPI does not provide enough fan-controlling capabilities for it to be interesting in our study anyways.

2.3. SpeedFan on Windows

SpeedFan is a closed-source utility for the Microsoft Windows family of operating systems that allows interested users to monitor several environmental characteristics of their personal computers. The utility provides a GUI, supports many families of hardware monitoring chipsets, and has an interface for controlling the duty cycle of the fans.

For our purposes, however, the utility has a fundamental flaw, in that it provides no interfacing for the automatic in-chip fan controlling. This means that whatever policy the user specifies in the utility, can be preserved only whilst the utility is still running, and if something happens either to the utility or the operating system, then the thermal characteristics of the system can no longer be predicted or maintained.

2.4. lm_sensors on Linux

The lm_sensors package is the most popular hardware monitoring package for the Linux kernel, supporting a variety of different hardware monitoring chips. However, the package is known to be difficult to configure even for otherwise experienced system administrators, and to our knowledge is not available on any BSD platform.

3. Hardware Monitoring Chips

In this section, we will overview some of the basics regarding the hardware part of the fan control, as well as provide an outline of some interesting functionalities that popular hardware monitoring chips implement.

First, let us start with the fans. Fans in the desktop computers are usually rated for 12 volts, although most would still run at 7 volts, where few would run when voltage is lower than 5 volts. Often, fans require higher voltage in order to start, as opposed to the voltage that would ensure that they continue running. It has also been observed that fans require higher voltage when their temperature is low, compared to the same fan when it is warm. If not accounted properly, this could have negative effects on system stability when, for example, the system is cold-started with a physically limiting fan-controlling solution, such as Zalman Fan Mate 2, set to the lowest voltage which, although sufficient for continued operation, may not be sufficient for a cold start, resulting in the fan making the clicking noises without actually ever spinning, causing a violation of the thermal characteristics of the box.

Our prototype implementation is concentrated on the Winbond Super I/O Hardware Monitors, which account for the bulk majority of the sensor chips that are available in popular motherboards; we will thus describe some functionality that is available in the said chips. Our initial patch implements support for the following three families of Winbond Super I/O chips: W83627HF, W83627THF / 37HF and W83627EHF / DHG.

Many of the hardware monitoring chips feature not only the manual mode, where the duty cycle of the fans could be changed directly, but also different types of cruising, in case of Winbond, this includes Thermal Cruise and Fan Speed Cruise, where the chip, once programmed with the set target temperature or target fan speed, internally determines what duty cycle should the fans be running at in order to satisfy the set cruising requirements.

For simplicity reasons, we have only implemented the manual mode and the thermal cruise mode in our initial prototype implementation. This is also partly due to the fact that the usefulness of the Fan Speed Cruise mode is somewhat questionable, as fans vary between each other, but within each fan it's not very likely that the speed will significantly alternate when the same voltage is given.

3.1. Shortcomings with general-purpose fan-control software

There is one problem that remains unavoidable with any general-purpose fan-controlling software: although many motherboards are in fact wired to do fan control, even if such features are not specifically advertised in motherboards' documentation or are available in the BIOS menus, some cheaper boards that do feature chips that support fan controlling simply do not have the chips wired appropriately with the fan connector headers, such that any attempts to control the speed of the fans from within the hardware monitoring chip may not be successful.

To save costs yet allow the user to still perform some fan control, some boards often feature such wiring that all the fan headers are wired and controlled through a single pin and setting of the fan-controlling chip, even if the chip itself does offer individual pins and controlling settings for more than a single fan.

Unfortunately, there is no known general approach that can reliably detect these situations in due course and with reasonably simple and straightforward logic, so the task of determining the exact peculiarities in supported fan-control functionalities of a mainboard in question are left for the end user to establish on their own.

4. OpenBSD sysctl hw.sensors Fan Control

In this section, we will briefly describe the general notions of the OpenBSD's sysctl hardware sensors framework [Murenin.IEEE07] [Murenin.Asia09], and then provide some suggestions on how it can be altered such as to provide interfacing to the fan-controlling functionalities of the hardware monitoring chips.

The underlying mechanism that is used to transport the datastructures of the hardware sensors framework in OpenBSD is the sysctl interface. Currently, the framework is implemented in such a way that it allows only the drivers to export the data into the sysctl tree, but not get any feedback back from the user. However, the changes that the framework requires in order to support the functionality of passing modified sensor values back from the userland to the kernel are rather minimal, as we will explore in the following paragraphs.

One of the ways to accomplish the required functionality is to allow the userland to simply pass the modified sensor data for those sensors which the drivers specifically identify as modifiable. To avoid overengineering, we can also make an assumption that only an integer value should have the possibility of being modified and passed back to the driver, as opposed to the whole sensor structure. With these assumptions, the required modifications for the framework are very straightforward and minimal, as one could see from the patch that we have released. [Murenin.tech09] The changes in the framework were made to */sys/sys/sensors.h*, */sys/kern/kern_sysctl.c* and *sbin/sysctl/sysctl.c*, and we will briefly describe the changes below.

4.1. New *upvalue* field and new flags

The */sys/sys/sensors.h*, which is the header file with the definitions of the structures required for the framework, hereby sees the introduction of the *upvalue* field inside the *struct ksensor* structure, as well as two new flags, *SENSOR_FCONTROLLABLE* and *SENSOR_FNEWVALUE*.

Note that we have introduced the new *upvalue* field only into the kernel version of the sensor structure — it was deemed unnecessary to introduce the field for the userland version of the structure, since *upvalue* is only intended as the input for the drivers, and then after it is consumed by the driver, the value that the user has set would not be something that the user should be interested in monitoring. The primary reason for allowing such a discrepancy between the kernel's *struct ksensor* and userland's *struct sensor* is that, unless omitted from the userland, the introduction of a new field will cause *sizeof(struct sensor)* to grow, and would thus break the ABI, where the existing C/C++ *sysctl(3)* applications that are trying to get the *struct sensor* structure would not allocate large enough buff-

ers for the *sysctl(3)* to copy out the structure from the kernel to the userland, returning an [ENOMEM] error message instead.

The *controllable* flag, when set by the driver, signifies that the sensor is a read/write sensor. After a new value is provided by the user, it is stored into the *upvalue* field, and the *newvalue* flag is set, which then remains set until the driver's periodic refresh procedure, which loops through all the sensors to make the necessary updates, consumes the sensor's *upvalue* and clears the flag.

It deserves mentioning that the way we have designed our fan controlling prototype is such that the driver never modifies any fan-controlling settings inside the chips unless the user explicitly requests any such changes. This has several benefits, one of which is that users should not feel intimidated that the mere fact of applying the patch and rebooting the system is going to do any damage to their system. In fact, there may be situations where the user might simply want to check on the policies the motherboard manufacturer has preloaded the chips with, and as our patch not only allows one to modify the existing fan-controlling behaviour, but also to monitor the currently applicable settings, or, at the very least, the duty cycle settings that the fans are experiencing, the user does indeed has such an option.

4.2. Sensor types

The next design decision that we would like to discuss is that of *sensor types*. Now that the drivers could declare that certain sensors could have an *upvalue* field that could be modified and passed back into the driver, the question regarding sensor types comes to mind. On the one hand, any new sensor type would break the ABI and, possibly, API of existing utilities, whereas if the existing sensor types are reused, the interfacing may seem to look a bit too generic and somewhat less user-friendly.

For example, if we reuse the *temp* type to specify target temperatures, then those target temperature setting sensors would have to be numbered in the same namespace as those sensors that report actual temperature readings, e.g. temp0 may be the actual temperature sensor, whereas temp3 would be the (corresponding or not) read/write target temperature setting sensor.

One problem that we have found with the approach of reusing the existing sensor types is that not all types appear to be represented in the current version of *sensors.h*. For example, one of the settings that we might want the user to be able to modify is the stop, step-down and step-up time, expressed in seconds, and although there is a sensor type *timedelta*, expressed as a time fraction, it appears that the current use exclusively suggests that the value of such *timedelta* sensors should show the difference between the local wall clock and the wall clock of some external and more accurate timesource. [Balmer.Asia07] [Balmer.Euro07] Therefore, one must be careful in reuse of such sensor types, as it may inadvertently confuse tools like ntpd, creating a situation where it could be using such a sensor to adjust the drift of the local clock for very unintended results.

Although introducing new sensor types is very straightforward (a matter of defining each type in two places inside the */sys/sys/sensors.h*, supporting the printout in sysctl, sensorsd, systat etc, and changing the respective sensors in the drivers to the new type), the approach that we have taken thus far in our prototype implementation is to delay any such introduction, allowing us to make an interesting observation that we have managed to implement the fan-controlling interfacing via sysctl *hw.sensors* tree without breaking neither the existing API nor even the ABI, with the new functionality introduced exclusively on top, but not in place of, any parts of the existing framework.

4.3. Dynamic sensor descriptions

Settings for certain writable sensors may sometimes be rather complicated; for example, the duty cycle of fans may be controlled through several ways, including one manual and several automatic modes. In order to show these settings, we have conveniently used the description field of relevant sensors, where, depending on the data in certain registers, we would update the string describing an individual sensor with the information regarding some complex settings of the said sensor.

4.4. The lm(4) driver

In our prototype, we have implemented fan-controlling support for several chips that are otherwise supported by OpenBSD's *lm(4)* driver. We will briefly describe what went on in our implementation.

First, as already mentioned, the driver doesn't modify the fan-controlling behaviour unless the user specifically requests such modifications via the sysctl interfacing.

Then, we have tried to make the interfacing as intuitive as possible. For example, when the user tries to modify the duty cycle of the fans directly through the

percent type sensors, the respective fan control settings automatically switch into the manual mode; same happens when the user tries to change the target temperature of a given fan-controlling pin.

Table I provides an example summary of what sensors were newly added to *lm(4)*, although the exact sensors will differ with each supported family. In the example below, you can see that the chips themselves in this particular family can independently control 4 fans (which is not to say that the motherboard manufacturer has necessarily wired everything to make such independent control possible). Note that all of these new sensors are both readable and writable.

Sensor	Usage
percent{0,1,2,3}	summary and dury cycle
temp{3,4,5,6}	target temperature
temp{7,8,9,10}	temperature tolerance
percent{4,5,6,7}	Start-up duty cycle
percent{8,9,10,11}	Stop duty cycle
indicator{0,1,2,3}	PWM/DC switch

Table I. Newly added sensors for the W83627EHF / DHG family.

5. Demonstration

We will hereby demonstrate some functionalities of our prototype, together with the relevant commentary.

Below is the output from a W83627DHG chip on an Intel D201GLY2 box with one small system fan. Values that are not applicable to the current operational mode are automatically marked as 'unknown' in sysctl.

```
% dmesg | fgrep W83627DHG
wbsio0 at isa0 port 0x4e/2: W83627DHG rev 0x25
lm1 at wbsio0 port 0x290/8: W83627DHG

% sysctl hw.sensors
hw.sensors.cpu0.temp0=58.00 degC
hw.sensors.lm1.temp0=45.00 degC (Sys)
hw.sensors.lm1.temp1=51.00 degC (CPU)
hw.sensors.lm1.temp2=14.50 degC (Aux)
hw.sensors.lm1.temp3=38.00 degC (Sys Target)
hw.sensors.lm1.temp4=unknown (CPU Target)
hw.sensors.lm1.temp5=unknown (Aux Target)
hw.sensors.lm1.temp6=unknown (CPU Target)
hw.sensors.lm1.temp7=2.00 degC (Sys Tolerance)
hw.sensors.lm1.temp8=unknown (CPU Tolerance)
hw.sensors.lm1.temp9=unknown (Aux Tolerance)
hw.sensors.lm1.temp10=unknown (CPU Tolerance)
hw.sensors.lm1.fan0=1854 RPM (Sys)
hw.sensors.lm1.volt0=1.34 VDC (VCore)
hw.sensors.lm1.volt1=12.20 VDC (+12V)
hw.sensors.lm1.volt2=3.33 VDC (+3.3V)
hw.sensors.lm1.volt3=3.33 VDC (+3.3V)
hw.sensors.lm1.volt4=-3.95 VDC (-12V)
hw.sensors.lm1.volt5=0.11 VDC
hw.sensors.lm1.volt6=1.62 VDC
hw.sensors.lm1.volt7=3.28 VDC (3.3VSB)
hw.sensors.lm1.volt8=0.03 VDC (VBAT)
hw.sensors.lm1.indicator0=Off (Sys Fan PWM/DC: PWM)
hw.sensors.lm1.indicator1=Off (CPU Fan PWM/DC: PWM)
hw.sensors.lm1.indicator2=Off (Aux Fan PWM/DC: PWM)
hw.sensors.lm1.indicator3=On (CPU Fan PWM/DC: DC)
hw.sensors.lm1.percent0=100.00% (Sys Fan PWM Thermal), OK
hw.sensors.lm1.percent1=100.00% (CPU Fan PWM Manual), OK
hw.sensors.lm1.percent2=100.00% (Aux Fan PWM Manual), OK
hw.sensors.lm1.percent3=100.00% (CPU Fan DC SmartIII), OK
hw.sensors.lm1.percent4=0.39% (Sys Fan Start-up Value), CRITICAL
hw.sensors.lm1.percent5=unknown (CPU Fan Start-up Value)
hw.sensors.lm1.percent6=unknown (Aux Fan Start-up Value)
hw.sensors.lm1.percent7=unknown (CPU Fan Start-up Value)
hw.sensors.lm1.percent8=29.41% (Sys Fan Stop Value), CRITICAL
hw.sensors.lm1.percent9=unknown (CPU Fan Stop Value)
hw.sensors.lm1.percent10=unknown (Aux Fan Stop Value)
hw.sensors.lm1.percent11=unknown (CPU Fan Stop Value)
```

We alter the target temperature value for the Thermal Cruise mode, and note that the percento value is going down.

```
% sudo sysctl hw.sensors.lm1.temp3=50
hw.sensors.lm1.temp3=38.00 degC {updating} (Sys Target)

% sysctl hw.sensors | fgrep Sys
hw.sensors.lm1.temp0=45.00 degC (Sys)
hw.sensors.lm1.temp3=50.00 degC (Sys Target)
hw.sensors.lm1.temp7=2.00 degC (Sys Tolerance)
hw.sensors.lm1.fan0=1739 RPM (Sys)
hw.sensors.lm1.indicator0=Off (Sys Fan PWM/DC: PWM)
hw.sensors.lm1.percent0=29.41% (Sys Fan PWM Thermal), CRITICAL
hw.sensors.lm1.percent4=0.39% (Sys Fan Start-up Value), CRITICAL
hw.sensors.lm1.percent8=29.41% (Sys Fan Stop Value), CRITICAL
```

Our D201GLY2 board is deemed abnormal, because the fan doesn't stop much until the duty cycle is almost zero. (Or, perhaps, the issue lies with the fan of the enclosure where the board resides.) So the result is likely to be entirely different on a different board; the status field indicates the likelihood that the fan is not going to run on a given duty cycle.

```
% sudo sysctl hw.sensors.lm1.percent8=10
hw.sensors.lm1.percent8=29.41% {updating} (Sys Fan Stop Value), CRITICAL

% sysctl hw.sensors | fgrep Sys
hw.sensors.lm1.temp0=45.00 degC (Sys)
hw.sensors.lm1.temp3=50.00 degC (Sys Target)
hw.sensors.lm1.temp7=2.00 degC (Sys Tolerance)
hw.sensors.lm1.fan0=1240 RPM (Sys)
hw.sensors.lm1.indicator0=Off (Sys Fan PWM/DC: PWM)
hw.sensors.lm1.percent0=9.80% (Sys Fan PWM Thermal), CRITICAL
hw.sensors.lm1.percent4=0.39% (Sys Fan Start-up Value), CRITICAL
hw.sensors.lm1.percent8=9.80% (Sys Fan Stop Value), CRITICAL
```

Now let's go into the Manual mode. Note that the description of the percento sensor will change to indicate that the Manual mode becomes active, and that the value goes gradually towards the desired value over some period of time.

```
% sudo sysctl hw.sensors.lm1.percent0=6
hw.sensors.lm1.percent0=9.80% {updating} (Sys Fan PWM Thermal), CRITICAL

% sysctl hw.sensors | fgrep Sys
hw.sensors.lm1.temp0=45.00 degC (Sys)
hw.sensors.lm1.temp3=unknown (Sys Target)
hw.sensors.lm1.temp7=unknown (Sys Tolerance)
hw.sensors.lm1.fan0=1240 RPM (Sys)
hw.sensors.lm1.indicator0=Off (Sys Fan PWM/DC: PWM)
hw.sensors.lm1.percent0=9.80% (Sys Fan PWM Manual), CRITICAL
hw.sensors.lm1.percent4=unknown (Sys Fan Start-up Value)
hw.sensors.lm1.percent8=unknown (Sys Fan Stop Value)

% sysctl hw.sensors | fgrep Sys
hw.sensors.lm1.temp0=45.00 degC (Sys)
hw.sensors.lm1.temp3=unknown (Sys Target)
hw.sensors.lm1.temp7=unknown (Sys Tolerance)
hw.sensors.lm1.fan0=781 RPM (Sys)
hw.sensors.lm1.indicator0=Off (Sys Fan PWM/DC: PWM)
hw.sensors.lm1.percent0=5.88% (Sys Fan PWM Manual), CRITICAL
hw.sensors.lm1.percent4=unknown (Sys Fan Start-up Value)
hw.sensors.lm1.percent8=unknown (Sys Fan Stop Value)
```

We would like to emphasise that the driver only implements reading and writing to the registers of the chip, e.g. the Thermal Cruise mode is still performed by the chip itself. Fan Cruise mode and the Smart Fan III modes are not supported, although one can still monitor their effects via the percent{0,1,2,3} sensors.

6. Conclusion

We have described some hardware monitoring and fan-controlling functionalities of modern chips and provided a prototype that allows users to conveniently interface with the fan-controlling functionalities of their commonplace hardware.

7. Future Projects

Several future projects remain possible in regards to the sensors framework and fan control. In this section, we will try to identify some of them. Some of these identified projects may be of immediate interest to the actual users of the described systems, whereas others may be of more interest to the researchers of the subject.

7.1. Even easier fan control

Fan-speed controlling was discussed in this paper, and a prototype has been provided, however, further research is possible in several distinct directions.

In general, as we have shown, OpenBSD's sensors framework requires very little amount of modification to provide an interface for the ability to conveniently pass values from *sysctl(8)* back into the driver, such that the driver, in turn, could pass such values down to the chip, for the chip to modify the voltage of some fan headers in a certain predetermined fashion.

However, different generations of chips have different logic regarding fan control; many chips of recent generations have multiple temperature levels at which different fan speeds could be sought; certain temperature sensors could be specified to affect decisions regarding the speed of certain fans etc. Concerns for simplicity extinction are amplified by the fact that the majority of motherboards are miswired as far as hardware monitoring datasheets are concerned, since many modern hardware monitoring chips oftentimes provide way much more functionality in regards to fan controlling than the motherboard manufacturer is usually interested in supporting and advertising in its products for its endusers. Therefore, a complete, flexible and round patch for supporting fan controlling functionality might be a long way from OpenBSD's philosophy of being a system where a great deal of effort is paid towards the simplicity and generality of its feature set.

Due to these reasons, it is unclear if any general-purpose fan-controlling prototype will ever be integrated into the main release of the OpenBSD system, so some further research is warranted.

7.2. Possible race conditions between user software and the BIOS

It has also been speculated that user-initiated intervention with the fan-controlling functionality of the chip may cause undesirable consequences to the stability of the system as a result of certain conflicts with the system management code of the BIOS. Although the concern has some grounds, in our experience no undesirable interactions were found as of yet in regards to the matter. Future work may examine assembly code and, perhaps, specifications of various hardware components to determine if the concerns have some more valid grounds.

Motherboard manufacturers may wish to provide fan-controlling specifications through custom ACPI devices, whereby the specifications would be more definitive and less likely to harm the stability and design of the system, at the same time ensuring that each BIOS contains the relevant information of which pins of the chip are actually utilised in the design of the motherboard.

7.3. Fan controlling through sensorsd

Future work may also be done in regards to a simplified language for specifying various relationships for fan control, and the language may feature fall backs for the `sensorsd` hardware monitoring daemon for those chips that cannot do the monitoring loop by themselves.

References

[Balmer.Asia07] Marc Balmer. "Support for Radio Clocks in OpenBSD". In: AsiaBSDCon 2007 Proceedings. 8–11 March 2007, Tokyo, Japan.
http://www.openbsd.org/papers/radio-clocks-asiabsdcon07.pdf

[Balmer.Euro07] Marc Balmer. "Supporting Radio Clocks in OpenBSD". On: EuroBSDCon 2007. 12–15 September 2007, Copenhagen, Denmark. Slides:
http://www.openbsd.org/papers/eurobsdcon07/mbalmer-radio_clocks.pdf

[Evans.stress] Gary W. Evans and Dana Johnson. "Stress and Open-Office Noise". Journal of Applied Psychology, vol. 85, no. 5, pp. 779–783. October 2000.
doi:10.1037/0021-9010.85.5.779

[Google.FAST07] Eduardo Pinheiro, Wolf-Dietrich Weber and Luiz Andre Barroso. "Failure Trends in a Large Disk Drive Population". Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST'07). February 2007, San Jose, CA, USA.
http://labs.google.com/papers/disk_failures.pdf

[Intel.ACPI] —. "Advanced Configuration and Power Interface". Hewlett-Packard, Intel, Microsoft, Phoenix and Toshiba. http://www.acpi.info/

[Intel.acpica] —. "ACPI Component Architecture". Intel. http://www.acpica.org/

[Intel.D201GLY2] —. "Intel® Desktop Board D201GLY2 / D201GLY2A". Intel.
http://www.intel.com/products/motherboard/D201GLY2/configs.htm

[Murenin.BSc06] Constantine A. Murenin, B. Sc. (Hons) Final Year Project Main Report: "Microprocessor system hardware monitors. Interfacing on OpenBSD with sysctl(8)." Faculty of Computing Sciences and Engineering, De Montfort University, Leicester, UK, May 2006.

[Murenin.TOJ06] Constantine A. Murenin. "New two-level sensor API". The OpenBSD Journal. 30 December 2006. http://undeadly.org/cgi?action=article&sid=20061230235005

[Murenin.IEEE07] Constantine A. Murenin. "Generalised Interfacing with Microprocessor System Hardware Monitors". In: Proceedings of 2007 IEEE International Conference on Networking, Sensing and Control. 15–17 April 2007, London, United Kingdom. IEEE ICNSC 2007, pp. 901—906. doi:10.1109/ICNSC.2007.372901

[Murenin.Asia09] Constantine A. Murenin and Raouf Boutaba. "OpenBSD Hardware Sensors Framework". In: AsiaBSDCon 2009 Proceedings. 12–15 March 2009, Tokyo University of Science, Tokyo, Japan.
http://www.openbsd.org/papers/asiabsdcon2009-sensors-paper.pdf

[Murenin.tech09] Constantine A. Murenin. "sysctl hw.sensors lm(4) fan-controlling prototype/hack". tech@openbsd.org mailing list. 8 May 2009.

[thinkwiki] —. "How to control fan speed". ThinkWiki. http://www.thinkwiki.org/wiki/How_to_control_fan_speed

[Watanabe.ACPI] Takanori Watanabe. "ACPI implementation on FreeBSD". 2002 USENIX Annual Technical Conference, FREENIX Track. 10–15 June 2002, Monterey, CA, USA.

Biography

Constantine A. Murenin a.k.a. cnst is an MMath graduate student at the David R. Cheriton School of Computer Science at the University of Waterloo (CA). Prior to his graduate appointment, Constantine attended and subsequently graduated from East Carolina University (US) and De Montfort University (UK), receiving two Bachelor of Science degrees in Computer Science, with Honors and Honours, in 2007 and 2006, respectively. Active member of multiple open-source projects and a Google Summer of Code Alumnus, Constantine's interests range from standards compliance and usability at all levels, to quiet computing and hardware monitoring. You can contact him via email at <*C++@Cns.SU*>.

Raouf Boutaba received the MSc and PhD degrees in Computer Science from the University Pierre and Marie Curie, Paris, France, in 1990 and 1994, respectively. He is currently a Professor of Computer Science at the University of Waterloo. His research interests include network, resource and service management in wired and wireless networks. He is the founder and Editor-in-Chief of the IEEE Transactions on Network and Service Management and is on the editorial boards of several other journals. He is currently a distinguished lecturer of the IEEE Communications Society, the chairman of the IEEE Technical Committee on Information Infrastructure and the Director of the ComSoc Related Societies Board. He has received several best paper awards and other recognitions such as the Premier's Research Excellence Award.