Generic Thermal Sysfs driver How To

Written by Sujith Thomas <sujith.thomas@intel.com>, Zhang Rui <rui.zhang@intel.com>

Updated: 2 January 2008

0. Introduction

The generic thermal sysfs provides a set of interfaces for thermal zone devices (sensors) and thermal cooling devices (fan, processor...) to register with the thermal management solution and to be a part of it.

This how-to focuses on enabling new thermal zone and cooling devices to participate in thermal management. This solution is platform independent and any type of thermal zone devices and cooling devices should be able to make use of the infrastructure.

The main task of the thermal sysfs driver is to expose thermal zone attributes as well as cooling device attributes to the user space. An intelligent thermal management application can make decisions based on inputs from thermal zone attributes (the current temperature and trip point temperature) and throttle appropriate devices.

- *[0-*]* denotes any positive number starting from 0
- *[1-*]* denotes any positive number starting from 1

1. thermal sysfs driver interface functions

1.1 thermal zone device interface

```
struct thermal_zone_device
*thermal_zone_device_register(char *type,
                              int trips, int mask, void *devdata,
                              struct thermal_zone_device_ops *ops,
                              const struct thermal_zone_params *tzp,
                              int passive_delay, int polling_delay))
```

This interface function adds a new thermal zone device (sensor) to /sys/class/thermal folder as *thermal_zone[0-*]*. It tries to bind all the thermal cooling devices registered at the same time.

type:

the thermal zone type.

trips:

the total number of trip points this thermal zone supports.

mask:

Bit string: If 'n'th bit is set, then trip point 'n' is writable.

devdata:

device private data

ops:

thermal zone device call-backs.

.bind:

bind the thermal zone device with a thermal cooling device.

.unbind:

unbind the thermal zone device with a thermal cooling device.

.get_temp:

get the current temperature of the thermal zone.

.set_trips:

set the trip points window. Whenever the current temperature is updated, the trip points immediately below and above the current temperature are found.

.get_mode:

get the current mode (enabled/disabled) of the thermal zone.

- "enabled" means the kernel thermal management is enabled.

- "disabled" will prevent kernel thermal driver action upon trip points so that user applications can take charge of thermal management.

.set_mode:

set the mode (enabled/disabled) of the thermal zone.

.get_trip_type:

get the type of certain trip point.

.get_trip_temp:

get the temperature above which the certain trip point will be fired.

.set_emul_temp:

set the emulation temperature which helps in debugging different threshold temperature points.

tzp:

thermal zone platform parameters.

passive_delay:

number of milliseconds to wait between polls when performing passive cooling.

polling_delay:

number of milliseconds to wait between polls when checking whether trip points have been crossed (0 for interrupt driven systems).

```
void thermal_zone_device_unregister(struct thermal_zone_device *tz)
```

This interface function removes the thermal zone device. It deletes the corresponding entry from /sys/class/thermal folder and unbinds all the thermal cooling devices it uses.

```
struct thermal_zone_device
*thermal_zone_of_sensor_register(struct device *dev, int sensor_id,
                    void *data,
                    const struct thermal_zone_of_device_ops *ops)
```

This interface adds a new sensor to a DT thermal zone. This function will search the list of thermal zones described in device tree and look for the zone that refer to the sensor device pointed by dev->of_node as temperature providers. For the zone pointing to the sensor node, the sensor will be added to the DT thermal zone device.

The parameters for this interface are:

dev:

Device node of sensor containing valid node pointer in dev->of_node.

sensor_id:

a sensor identifier, in case the sensor IP has more than one sensors

data:

a private pointer (owned by the caller) that will be passed back, when a temperature reading is needed.

ops:

struct thermal_zone_of_device_ops *.

get_temp	a pointer to a function that reads the sensor temperature. This is mandatory callback provided by sensor driver.
set_trips	a pointer to a function that sets a temperature window. When this window is left the driver must inform the thermal core via thermal_zone_device_update.
get_trend	a pointer to a function that reads the sensor temperature trend.
set_emul_temp	a pointer to a function that sets sensor emulated temperature.

The thermal zone temperature is provided by the get_temp() function pointer of thermal_zone_of_device_ops. When called, it will have the private pointer @data back.

It returns error pointer if fails otherwise valid thermal zone device handle. Caller should check the return handle with IS_ERR() for finding whether success or not.

```
void thermal_zone_of_sensor_unregister(struct device *dev,
                                       struct thermal_zone_device *tzd)
```

This interface unregisters a sensor from a DT thermal zone which was successfully added by interface thermal_zone_of_sensor_register(). This function removes the sensor callbacks and private data from the thermal zone device registered with thermal_zone_of_sensor_register() interface. It will also silent the zone by remove the .get_temp() and get_trend() thermal zone device callbacks.

```
struct thermal_zone_device
*devm_thermal_zone_of_sensor_register(struct device *dev,
                      int sensor_id,
                      void *data,
                      const struct thermal_zone_of_device_ops *ops)
```

This interface is resource managed version of thermal_zone_of_sensor_register().

All details of thermal_zone_of_sensor_register() described in section 1.1.3 is applicable here.

The benefit of using this interface to register sensor is that it is not require to explicitly call thermal_zone_of_sensor_unregister() in error path or during driver unbinding as this is done by driver resource manager.

```
void devm_thermal_zone_of_sensor_unregister(struct device *dev,
                                struct thermal_zone_device *tzd)
```

This interface is resource managed version of thermal_zone_of_sensor_unregister(). All details of thermal_zone_of_sensor_unregister() described in section 1.1.4 is applicable here. Normally this function will not need to be called and the resource management code will ensure that the resource is freed.

```
int thermal_zone_get_slope(struct thermal_zone_device *tz)
```

This interface is used to read the slope attribute value for the thermal zone device, which might be useful for platform drivers for temperature calculations.

```
int thermal_zone_get_offset(struct thermal_zone_device *tz)
```

This interface is used to read the offset attribute value for the thermal zone device, which might be useful for platform drivers for temperature calculations.

1.2 thermal cooling device interface

```
struct thermal_cooling_device
*thermal_cooling_device_register(char *name,
                void *devdata, struct thermal_cooling_device_ops *)
```

This interface function adds a new thermal cooling device (fan/processor/...) to /sys/class/thermal/ folder as *cooling_device[0-*]*. It tries to bind itself to all the thermal zone devices registered at the same time.

name:

the cooling device name.

devdata:

device private data.

ops:

thermal cooling devices call-backs.

.get_max_state:
get the Maximum throttle state of the cooling device.
.get_cur_state:
get the Currently requested throttle state of the cooling device.
.set_cur_state:
set the Current throttle state of the cooling device.

```
void thermal_cooling_device_unregister(struct thermal_cooling_device *cdev)
```

This interface function removes the thermal cooling device. It deletes the corresponding entry from /sys/class/thermal folder and unbinds itself from all the thermal zone devices using it.

1.3 interface for binding a thermal zone device with a thermal cooling device

```
int thermal_zone_bind_cooling_device(struct thermal_zone_device *tz,
        int trip, struct thermal_cooling_device *cdev,
        unsigned long upper, unsigned long lower, unsigned int weight);
```

This interface function binds a thermal cooling device to a particular trip point of a thermal zone device.

This function is usually called in the thermal zone device .bind callback.

tz:
: the thermal zone device

cdev:
: thermal cooling device

trip:
: indicates which trip point in this thermal zone the cooling device is associated with.

upper:
: the Maximum cooling state for this trip point. THERMAL_NO_LIMIT means no upper limit, and the cooling device can be in max_state.

lower:
: the Minimum cooling state can be used for this trip point. THERMAL_NO_LIMIT means no lower limit, and the cooling device can be in cooling state 0.

weight:
: the influence of this cooling device in this thermal zone. See 1.4.1 below for more information.

```
int thermal_zone_unbind_cooling_device(struct thermal_zone_device *tz,
                        int trip, struct thermal_cooling_device *cdev);
```

This interface function unbinds a thermal cooling device from a particular trip point of a thermal zone device. This function is usually called in the thermal zone device .unbind callback.

tz:
: the thermal zone device

cdev:
: thermal cooling device

trip:
: indicates which trip point in this thermal zone the cooling device is associated with.

1.4 Thermal Zone Parameters

```
struct thermal_bind_params
```

This structure defines the following parameters that are used to bind a zone with a cooling device for a particular trip point.

.cdev:
: The cooling device pointer

.weight:
: The 'influence' of a particular cooling device on this zone. This is relative to the rest of the cooling devices. For example, if all cooling devices have a weight of 1, then they all contribute the same. You can use percentages if you want, but it's not mandatory. A weight of 0 means that this cooling device doesn't contribute to the cooling of this zone unless all cooling devices have a weight of 0. If all weights are 0, then they all contribute the same.

.trip_mask:
: This is a bit mask that gives the binding relation between this thermal zone and cdev, for a particular trip point. If nth bit is set, then the cdev and thermal zone are bound for trip point n.

.binding_limits:
: This is an array of cooling state limits. Must have exactly 2 * thermal_zone.number_of_trip_points. It is an array consisting of tuples <lower-state upper-state> of state limits. Each trip will be associated with one state limit tuple when binding. A NULL pointer means <THERMAL_NO_LIMITS THERMAL_NO_LIMITS> on all trips. These limits are used when binding a cdev to a trip point.

.match:
: This call back returns success(0) if the 'tz and cdev' need to be bound, as per platform data.

```
struct thermal_zone_params
```

This structure defines the platform level parameters for a thermal zone. This data, for each thermal zone should come from the platform layer. This is an optional feature where some platforms can choose not to provide this data.

.governor_name:
: Name of the thermal governor used for this zone

.no_hwmon:
: a boolean to indicate if the thermal to hwmon sysfs interface is required. when no_hwmon == false, a hwmon sysfs interface will be created. when no_hwmon == true, nothing will be done. In case the thermal_zone_params is NULL, the hwmon interface will be created (for backward compatibility).

.num_tbps:
: Number of thermal_bind_params entries for this zone

.tbp:
: thermal_bind_params entries

2. sysfs attributes structure

RO	read only value
WO	write only value
RW	read/write value

Thermal sysfs attributes will be represented under /sys/class/thermal. Hwmon sysfs I/F extension is also available under /sys/class/hwmon if hwmon is compiled in or built as a module.

Thermal zone device sys I/F, created once it's registered:

```
/sys/class/thermal/thermal_zone[0-*]:
  |---type:                   Type of the thermal zone
  |---temp:                   Current temperature
  |---mode:                   Working mode of the thermal zone
  |---policy:                 Thermal governor used for this zone
  |---available_policies:     Available thermal governors for this zone
  |---trip_point_[0-*]_temp:  Trip point temperature
  |---trip_point_[0-*]_type:  Trip point type
  |---trip_point_[0-*]_hyst:  Hysteresis value for this trip point
  |---emul_temp:              Emulated temperature set node
  |---sustainable_power:      Sustainable dissipatable power
  |---k_po:                   Proportional term during temperature overshoot
  |---k_pu:                   Proportional term during temperature undershoot
  |---k_i:                    PID's integral term in the power allocator gov
  |---k_d:                    PID's derivative term in the power allocator
  |---integral_cutoff:        Offset above which errors are accumulated
  |---slope:                  Slope constant applied as linear extrapolation
  |---offset:                 Offset constant applied as linear extrapolation
```

Thermal cooling device sys I/F, created once it's registered:

```
/sys/class/thermal/cooling_device[0-*]:
  |---type:                   Type of the cooling device(processor/fan/...)
  |---max_state:              Maximum cooling state of the cooling device
  |---cur_state:              Current cooling state of the cooling device
  |---stats:                  Directory containing cooling device's statistics
  |---stats/reset:            Writing any value resets the statistics
  |---stats/time_in_state_ms: Time (msec) spent in various cooling states
  |---stats/total_trans:      Total number of times cooling state is changed
  |---stats/trans_table:      Cooling state transition table
```

Then next two dynamic attributes are created/removed in pairs. They represent the relationship between a thermal zone and its associated cooling device. They are created/removed for each successful execution of thermal_zone_bind_cooling_device/thermal_zone_unbind_cooling_device.

```
/sys/class/thermal/thermal_zone[0-*]:
  |---cdev[0-*]:              [0-*]th cooling device in current thermal zone
  |---cdev[0-*]_trip_point:   Trip point that cdev[0-*] is associated with
  |---cdev[0-*]_weight:       Influence of the cooling device in
                              this thermal zone
```

Besides the thermal zone device sysfs I/F and cooling device sysfs I/F, the generic thermal driver also creates a hwmon sysfs I/F for each _type_ of thermal zone device. E.g. the generic thermal driver registers one hwmon class device and build the associated hwmon sysfs I/F for all the registered ACPI thermal zones.

Please read Documentation/ABI/testing/sysfs-class-thermal for thermal zone and cooling device attribute details.

```
/sys/class/hwmon/hwmon[0-*]:
  |---name:                   The type of the thermal zone devices
  |---temp[1-*]_input:        The current temperature of thermal zone [1-*]
  |---temp[1-*]_critical:     The critical trip point of thermal zone [1-*]
```

Please read Documentation/hwmon/sysfs-interface.rst for additional information.

3. A simple implementation

ACPI thermal zone may support multiple trip points like critical, hot, passive, active. If an ACPI thermal zone supports critical, passive, active[0] and active[1] at the same time, it may register itself as a thermal_zone_device (thermal_zone1) with 4 trip points in all. It has one processor and one fan, which are both registered as thermal_cooling_device. Both are considered to have the same effectiveness in cooling the thermal zone.

If the processor is listed in _PSL method, and the fan is listed in _AL0 method, the sys I/F structure will be built like this:

```
/sys/class/thermal:
 |thermal_zone1:
    |---type:                   acpitz
    |---temp:                   37000
    |---mode:                   enabled
    |---policy:                 step_wise
    |---available_policies:     step_wise fair_share
```

```
    |---trip_point_0_temp:      100000
    |---trip_point_0_type:      critical
    |---trip_point_1_temp:      80000
    |---trip_point_1_type:      passive
    |---trip_point_2_temp:      70000
    |---trip_point_2_type:      active0
    |---trip_point_3_temp:      60000
    |---trip_point_3_type:      active1
    |---cdev0:                  --->/sys/class/thermal/cooling_device0
    |---cdev0_trip_point:       1       /* cdev0 can be used for passive */
    |---cdev0_weight:           1024
    |---cdev1:                  --->/sys/class/thermal/cooling_device3
    |---cdev1_trip_point:       2       /* cdev1 can be used for active[0]*/
    |---cdev1_weight:           1024

  |cooling_device0:
    |---type:                   Processor
    |---max_state:              8
    |---cur_state:              0

  |cooling_device3:
    |---type:                   Fan
    |---max_state:              2
    |---cur_state:              0

/sys/class/hwmon:
 |hwmon0:
    |---name:                   acpitz
    |---temp1_input:            37000
    |---temp1_crit:             100000
```

4. Export Symbol APIs

4.1. get_tz_trend

This function returns the trend of a thermal zone, i.e the rate of change of temperature of the thermal zone. Ideally, the thermal sensor drivers are supposed to implement the callback. If they don't, the thermal framework calculated the trend by comparing the previous and the current temperature values.

4.2. get_thermal_instance

This function returns the thermal_instance corresponding to a given {thermal_zone, cooling_device, trip_point} combination. Returns NULL if such an instance does not exist.

4.3. thermal_cdev_update

This function serves as an arbitrator to set the state of a cooling device. It sets the cooling device to the deepest cooling state if possible.

5. thermal_emergency_poweroff

On an event of critical trip temperature crossing the thermal framework shuts down the system by calling hw_protection_shutdown(). The hw_protection_shutdown() first attempts to perform an orderly shutdown but accepts a delay after which it proceeds doing a forced power-off or as last resort an emergency_restart.

The delay should be carefully profiled so as to give adequate time for orderly poweroff.

If the delay is set to 0 emergency poweroff will not be supported. So a carefully profiled non-zero positive value is a must for emergency poweroff to be triggered.