The Linux Hardware Monitoring kernel API

Guenter Roeck

Introduction

This document describes the API that can be used by hardware monitoring drivers that want to use the hardware monitoring framework.

This document does not describe what a hardware monitoring (hwmon) Driver or Device is. It also does not describe the API which can be used by user space to communicate with a hardware monitoring device. If you want to know this then please read the following file: Documentation/hwmon/sysfs-interface.rst.

For additional guidelines on how to write and improve hwmon drivers, please also read Documentation/hwmon/submitting-patches.rst.

The API

Each hardware monitoring driver must #include <linux/hwmon.h> and, in most cases, <linux/hwmon-sysfs.h>. linux/hwmon.h declares the following register/unregister functions:

```
struct device *
hwmon_device_register_with_groups(struct device *dev, const char *name,
                                  void *drvdata,
                                  const struct attribute_group **groups);

struct device *
devm_hwmon_device_register_with_groups(struct device *dev,
                                       const char *name, void *drvdata,
                                       const struct attribute_group **groups);

struct device *
hwmon_device_register_with_info(struct device *dev,
                                const char *name, void *drvdata,
                                const struct hwmon_chip_info *info,
                                const struct attribute_group **extra_groups);

struct device *
devm_hwmon_device_register_with_info(struct device *dev,
                                     const char *name,
                                     void *drvdata,
                                     const struct hwmon_chip_info *info,
                                     const struct attribute_group **extra_groups);

void hwmon_device_unregister(struct device *dev);

void devm_hwmon_device_unregister(struct device *dev);
```

hwmon_device_register_with_groups registers a hardware monitoring device. The first parameter of this function is a pointer to the parent device. The name parameter is a pointer to the hwmon device name. The registration function wil create a name sysfs attribute pointing to this name. The drvdata parameter is the pointer to the local driver data. hwmon_device_register_with_groups will attach this pointer to the newly allocated hwmon device. The pointer can be retrieved by the driver using dev_get_drvdata() on the hwmon device pointer. The groups parameter is a pointer to a list of sysfs attribute groups. The list must be NULL terminated. hwmon_device_register_with_groups creates the hwmon device with name attribute as well as all sysfs attributes attached to the hwmon device. This function returns a pointer to the newly created hardware monitoring device or PTR_ERR for failure.

devm_hwmon_device_register_with_groups is similar to hwmon_device_register_with_groups. However, it is device managed, meaning the hwmon device does not have to be removed explicitly by the removal function.

hwmon_device_register_with_info is the most comprehensive and preferred means to register a hardware monitoring device. It creates the standard sysfs attributes in the hardware monitoring core, letting the driver focus on reading from and writing to the chip instead of having to bother with sysfs attributes. The parent device parameter cannot be NULL with non-NULL chip info. Its parameters are described in more detail below.

devm_hwmon_device_register_with_info is similar to hwmon_device_register_with_info. However, it is device managed, meaning the hwmon device does not have to be removed explicitly by the removal function.

hwmon_device_unregister deregisters a registered hardware monitoring device. The parameter of this function is the pointer to the registered hardware monitoring device structure. This function must be called from the driver remove function if the hardware monitoring device was registered with hwmon_device_register_with_groups or hwmon_device_register_with_info.

devm_hwmon_device_unregister does not normally have to be called. It is only needed for error handling, and only needed if the driver probe fails after the call to devm_hwmon_device_register_with_groups or hwmon_device_register_with_info and if the

automatic (device managed) removal would be too late.

All supported hwmon device registration functions only accept valid device names. Device names including invalid characters (whitespace, '*', or '-') will be rejected. The 'name' parameter is mandatory.

Using devm_hwmon_device_register_with_info()

hwmon_device_register_with_info() registers a hardware monitoring device. The parameters to this function are

*struct device *dev*	Pointer to parent device
*const char *name*	Device name
*void *drvdata*	Driver private data
*const struct hwmon_chip_info *info*	Pointer to chip description.
*const struct attribute_group **extra_groups*	Null-terminated list of additional non-standard sysfs attribute groups.

This function returns a pointer to the created hardware monitoring device on success and a negative error code for failure.

The hwmon_chip_info structure looks as follows:

```
struct hwmon_chip_info {
        const struct hwmon_ops *ops;
        const struct hwmon_channel_info **info;
};
```

It contains the following fields:

- ops:
  Pointer to device operations.
- info:
  NULL-terminated list of device channel descriptors.

The list of hwmon operations is defined as:

```
struct hwmon_ops {
      umode_t (*is_visible)(const void *, enum hwmon_sensor_types type,
                            u32 attr, int);
      int (*read)(struct device *, enum hwmon_sensor_types type,
                  u32 attr, int, long *);
      int (*write)(struct device *, enum hwmon_sensor_types type,
                   u32 attr, int, long);
};
```

It defines the following operations.

- is_visible:
  Pointer to a function to return the file mode for each supported attribute. This function is mandatory.
- read:
  Pointer to a function for reading a value from the chip. This function is optional, but must be provided if any readable attributes exist.
- write:
  Pointer to a function for writing a value to the chip. This function is optional, but must be provided if any writeable attributes exist.

Each sensor channel is described with struct hwmon_channel_info, which is defined as follows:

```
struct hwmon_channel_info {
        enum hwmon_sensor_types type;
        u32 *config;
};
```

It contains following fields:

- type:
  The hardware monitoring sensor type.

  Supported sensor types are

hwmon_chip	A virtual sensor type, used to describe attributes which are not bound to a specific input or output
hwmon_temp	Temperature sensor
hwmon_in	Voltage sensor
hwmon_curr	Current sensor
hwmon_power	Power sensor

hwmon_energy	Energy sensor
hwmon_humidity	Humidity sensor
hwmon_fan	Fan speed sensor
hwmon_pwm	PWM control

- config:

  Pointer to a 0-terminated list of configuration values for each sensor of the given type. Each value is a combination of bit values describing the attributes supposed by a single sensor.

As an example, here is the complete description file for a LM75 compatible sensor chip. The chip has a single temperature sensor. The driver wants to register with the thermal subsystem (HWMON_C_REGISTER_TZ), and it supports the update_interval attribute (HWMON_C_UPDATE_INTERVAL). The chip supports reading the temperature (HWMON_T_INPUT), it has a maximum temperature register (HWMON_T_MAX) as well as a maximum temperature hysteresis register (HWMON_T_MAX_HYST):

```
static const u32 lm75_chip_config[] = {
        HWMON_C_REGISTER_TZ | HWMON_C_UPDATE_INTERVAL,
        0
};

static const struct hwmon_channel_info lm75_chip = {
        .type = hwmon_chip,
        .config = lm75_chip_config,
};

static const u32 lm75_temp_config[] = {
        HWMON_T_INPUT | HWMON_T_MAX | HWMON_T_MAX_HYST,
        0
};

static const struct hwmon_channel_info lm75_temp = {
        .type = hwmon_temp,
        .config = lm75_temp_config,
};

static const struct hwmon_channel_info *lm75_info[] = {
        &lm75_chip,
        &lm75_temp,
        NULL
};

The HWMON_CHANNEL_INFO() macro can and should be used when possible.
With this macro, the above example can be simplified to

static const struct hwmon_channel_info *lm75_info[] = {
        HWMON_CHANNEL_INFO(chip,
                           HWMON_C_REGISTER_TZ | HWMON_C_UPDATE_INTERVAL),
        HWMON_CHANNEL_INFO(temp,
                           HWMON_T_INPUT | HWMON_T_MAX | HWMON_T_MAX_HYST),
        NULL
};

The remaining declarations are as follows.

static const struct hwmon_ops lm75_hwmon_ops = {
        .is_visible = lm75_is_visible,
        .read = lm75_read,
        .write = lm75_write,
};

static const struct hwmon_chip_info lm75_chip_info = {
        .ops = &lm75_hwmon_ops,
        .info = lm75_info,
};
```

A complete list of bit values indicating individual attribute support is defined in include/linux/hwmon.h. Definition prefixes are as follows.

HWMON_C_xxxx	Chip attributes, for use with hwmon_chip.
HWMON_T_xxxx	Temperature attributes, for use with hwmon_temp.
HWMON_I_xxxx	Voltage attributes, for use with hwmon_in.
HWMON_C_xxxx	Current attributes, for use with hwmon_curr. Notice the prefix overlap with chip attributes.
HWMON_P_xxxx	Power attributes, for use with hwmon_power.
HWMON_E_xxxx	Energy attributes, for use with hwmon_energy.
HWMON_H_xxxx	Humidity attributes, for use with hwmon_humidity.

HWMON_F_xxxx	Fan speed attributes, for use with hwmon_fan.
HWMON_PWM_xxxx	PWM control attributes, for use with hwmon_pwm.

Driver callback functions

Each driver provides is_visible, read, and write functions. Parameters and return values for those functions are as follows:

```
umode_t is_visible_func(const void *data, enum hwmon_sensor_types type,
                        u32 attr, int channel)
```

Parameters:

data:
: Pointer to device private data structure.

type:
: The sensor type.

attr:
: Attribute identifier associated with a specific attribute. For example, the attribute value for HWMON_T_INPUT would be hwmon_temp_input. For complete mappings of bit fields to attribute values please see include/linux/hwmon.h.

channel:
: The sensor channel number.

Return value:
: The file mode for this attribute. Typically, this will be 0 (the attribute will not be created), S_IRUGO, or 'S_IRUGO | S_IWUSR'.

```
int read_func(struct device *dev, enum hwmon_sensor_types type,
              u32 attr, int channel, long *val)
```

Parameters:

dev:
: Pointer to the hardware monitoring device.

type:
: The sensor type.

attr:
: Attribute identifier associated with a specific attribute. For example, the attribute value for HWMON_T_INPUT would be hwmon_temp_input. For complete mappings please see include/linux/hwmon.h.

channel:
: The sensor channel number.

val:
: Pointer to attribute value.

Return value:
: 0 on success, a negative error number otherwise.

```
int write_func(struct device *dev, enum hwmon_sensor_types type,
               u32 attr, int channel, long val)
```

Parameters:

dev:
: Pointer to the hardware monitoring device.

type:
: The sensor type.

attr:
: Attribute identifier associated with a specific attribute. For example, the attribute value for HWMON_T_INPUT would be hwmon_temp_input. For complete mappings please see include/linux/hwmon.h.

channel:
: The sensor channel number.

val:
: The value to write to the chip.

Return value:
: 0 on success, a negative error number otherwise.

Driver-provided sysfs attributes

If the hardware monitoring device is registered with hwmon_device_register_with_info or devm_hwmon_device_register_with_info, it is most likely not necessary to provide sysfs attributes. Only additional non-standard sysfs attributes need to be provided when one of those registration functions is used.

The header file linux/hwmon-sysfs.h provides a number of useful macros to declare and use hardware monitoring sysfs attributes.

In many cases, you can use the exsting define DEVICE_ATTR or its variants DEVICE_ATTR_{RW,RO,WO} to declare such attributes. This is feasible if an attribute has no additional context. However, in many cases there will be additional information such as a sensor index which will need to be passed to the sysfs attribute handling function.

SENSOR_DEVICE_ATTR and SENSOR_DEVICE_ATTR_2 can be used to define attributes which need such additional context information. SENSOR_DEVICE_ATTR requires one additional argument, SENSOR_DEVICE_ATTR_2 requires two.

Simplified variants of SENSOR_DEVICE_ATTR and SENSOR_DEVICE_ATTR_2 are available and should be used if standard attribute permissions and function names are feasible. Standard permissions are 0644 for SENSOR_DEVICE_ATTR[_2]_RW, 0444 for SENSOR_DEVICE_ATTR[_2]_RO, and 0200 for SENSOR_DEVICE_ATTR[_2]_WO. Standard functions, similar to DEVICE_ATTR_{RW,RO,WO}, have _show and _store appended to the provided function name.

SENSOR_DEVICE_ATTR and its variants define a struct sensor_device_attribute variable. This structure has the following fields:

```
struct sensor_device_attribute {
        struct device_attribute dev_attr;
        int index;
};
```

You can use to_sensor_dev_attr to get the pointer to this structure from the attribute read or write function. Its parameter is the device to which the attribute is attached.

SENSOR_DEVICE_ATTR_2 and its variants define a struct sensor_device_attribute_2 variable, which is defined as follows:

```
struct sensor_device_attribute_2 {
        struct device_attribute dev_attr;
        u8 index;
        u8 nr;
};
```

Use to_sensor_dev_attr_2 to get the pointer to this structure. Its parameter is the device to which the attribute is attached.