hwmon(Hardware Monitoring)顾名思义，是linux为访问温度/电压/电流等硬件监控组件所提供的一个通用子系统，为应用层提供了标准的接口。hwmon利用sysfs实现，hwmon注册成功后，我们可以很方便的在/sys/class/hwmon/hwmon\*/目录查看或设置硬件监控器件参数。

# API说明

注册一个硬件监控设备。

struct device \* devm\_hwmon\_device\_register\_with\_groups(struct device \*dev, const char \*name, void \*drvdata, const struct attribute\_group \*\*groups);

dev：父设备

name：hwmon设备名称

drvdata：私有数据

groups：指向sysfs属性组列表的指针，列表需以NULL为结尾。

注册一个硬件监控设备，此方法比上面的注册方法更加的全面，我们应该优先使用。此函数会根据hwmon\_chip\_info创建标准的sysfs属性文件，当有非标准的sysfs属性时，我们可以通过传入extra\_groups创建自定义的sysfs属性文件。

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);

info：芯片的操作方法与配置信息

如果不使用devm函数进行设备注册，则需要释放。

void hwmon\_device\_unregister(struct device \*dev);

所以我们需要正确填充struct hwmon\_chip\_info以及struct attribute\_group才能正确创建sysfs属性文件。

## 传感器类型与属性

填充struct hwmon\_chip\_info时需要填充传感器类型与对应属性，hwmon.h中定义了目前支持的可用于创建标准的sysfs属性文件的hwmon传感器类型与类型下的各种属性：

enum hwmon\_sensor\_types {

hwmon\_chip,//虚拟类型，用于描述不属于下面这些类型的一些属性

hwmon\_temp,//温度

hwmon\_in,//电压

hwmon\_curr,//电流

hwmon\_power,//功率

hwmon\_energy,//能量

hwmon\_humidity,//湿度

hwmon\_fan,//风扇转速

hwmon\_pwm,//pwm

};

每一种类型都定义了很多不同的属性，下面对部分属性做出说明：

enum hwmon\_chip\_attributes {//虚拟传感器的属性

hwmon\_chip\_update\_interval,//芯片更新读数的时间间隔

};

enum hwmon\_temp\_attributes {//温度传感器的属性

hwmon\_temp\_input = 0,//温度值

hwmon\_temp\_type,//温度传感器类型

hwmon\_temp\_min,//温度最小值

hwmon\_temp\_min\_hyst,//最小温度滞后值

hwmon\_temp\_max,//温度最大值

hwmon\_temp\_max\_hyst,//最大温度滞后值

};

添加属性后，hwmon就会帮助我们创建标准的sysfs属性文件，Documentation\hwmon\sysfs-interface.rst中对这些属性文件含义做出了解释，属性文件名称具有标准性可以方便使用一些标准库。从文档中的属性文件含义可以推出hwmon.h中定义的各种属性的含义，比如添加了hwmon\_temp\_input属性后，就会创建temp[1-\*]\_input文件，其含义是当前温度值。

属性文件命名方案一般为：<type><number>\_<item>，type有temp(温度)、in(电压)等，item有input(测量值)、max(最大值)、min(最小值)等，number一般从1开始，但是电压是从0开始。

sysfs属性值格式为字符串，如果将数字写入则会被解释为0。

## hwmon\_chip\_info

芯片的操作方法与配置信息：

struct hwmon\_chip\_info {

const struct hwmon\_ops \*ops;//设备操作方法

const struct hwmon\_channel\_info \*\*info;//通道参数信息列表，列表以NULL为结尾

};

hwmon设备操作方法：

struct hwmon\_ops {

umode\_t (\*is\_visible)(const void \*drvdata, enum hwmon\_sensor\_types type, u32 attr, int channel);

int (\*read)(struct device \*dev, enum hwmon\_sensor\_types type,

u32 attr, int channel, long \*val);

int (\*write)(struct device \*dev, enum hwmon\_sensor\_types type,

u32 attr, int channel, long val);

};

.is\_visible：此函数必须提供，函数返回属性是否可见，也就是是否创建对应的sysfs属性文件。

drvdata：驱动私有数据

type：传感器类型

attr：传感器属性

channel：通道号

函数返回0则不会创建对应属性文件，非0时则根据返回的文件权限创建文件， 文件权限定义于stat.h中，下面对部分权限做出解释：

#define S\_IRWXU 00700//用户可读写执行

#define S\_IRUSR 00400//用户可读

#define S\_IWUSR 00200//用户可写

#define S\_IXUSR 00100//用户可执行

#define S\_IRWXG 00070//用户组可读写执行

#define S\_IRGRP 00040//用户组可读

#define S\_IWGRP 00020//用户组可写

#define S\_IXGRP 00010//用户组可执行

#define S\_IRWXO 00007//其他可读写执行

#define S\_IROTH 00004//其他可读

#define S\_IWOTH 00002//其他可写

#define S\_IXOTH 00001//其他可执行

.read：用于从芯片中读取数据，如果有可读属性则此函数必须提供。

.write：用于向芯片中写入数据，如果有可写属性则此函数必须提供。

hwmin通道参数信息：

struct hwmon\_channel\_info {

enum hwmon\_sensor\_types type;//通道传感器类型

const u32 \*config;//通道传感器属性，以NULL为结尾

};

.type：传感器类型，上面已讲。

.config：传感器属性，注意此属性按位表示不同含义，如需添加hwmon\_temp\_input和hwmon\_temp\_max属性，则config应该配置成HWMON\_T\_INPUT | HWMON\_T\_MAX，HWMON\_T\_INPUT为BIT(hwmon\_temp\_input)，定义都在hwmon.h中。

## attribute\_group

如果器件含有非上述介绍的标准sysfs属性的属性，才需要填充此结构，属性组的使用方法请参考sysfs文档。

我们在sysfs的文档中利用了struct kobj\_attribute完成了属性文件的读写，但是其实还有很多结构包含了struct attribute结构，如struct device\_attribute：

struct device\_attribute {

struct attribute attr;

ssize\_t (\*show)(struct device \*dev, struct device\_attribute \*attr,char \*buf);

ssize\_t (\*store)(struct device \*dev, struct device\_attribute \*attr,const char \*buf, size\_t count);

};

注意其show/store方法的第一个参数为dev，其指向我们调用hwmon设备注册函数返回的hwmon device，所以利用dev\_get\_drvdata(dev)可以很方便的获取我们在注册hwmon时传入的私有数据。

如果传感器的属性比较单一，我们利用struct device\_attribute就足够了，我们可以利用DEVICE\_ATTR()宏完成结构的初始化，然后添加到属性组中。但是当传感器属性比较多，比如含有多个相同类型的属性，如器件可以读取多个温度值(当然温度可以使用标准方法注册，只是用于举例)，我们无需为每个温度的读取创建方法，那么在读取方法中就要区分到底是读取的哪个温度，此时可以借用hwmon-sysfs.h提供的结构：

struct sensor\_device\_attribute{

struct device\_attribute dev\_attr;

int index;

};

利用SENSOR\_DEVICE\_ATTR()宏完成结构初始化，传入不同index区分相同类型的属性，读取/写入方法的参数有struct device\_attribute \*devattr，可以利用to\_sensor\_dev\_attr(devattr)得到struct sensor\_device\_attribute，然后通过index区分出到底想要读取哪个温度。

如果我们还想在读取或写入方法中获取更多参数来区分不同属性，还可以借用如下结构：

struct sensor\_device\_attribute\_2 {

struct device\_attribute dev\_attr;

u8 index;

u8 nr;

};

只是多出了nr参数，可以让我们填充额外参数，此结构利用SENSOR\_DEVICE\_ATTR\_2()宏完成初始化，利用to\_sensor\_dev\_attr\_2(devattr)可以获取struct sensor\_device\_attribute\_2，然后就可以使用index/nr参数了。

# 标准库使用

得益于hwmon用户层接口的标准化，可以使用标准库lm-sensors来读取可用传感器。

获取：

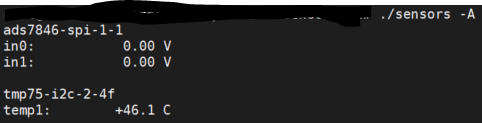
<https://github.com/lm-sensors/lm-sensors.git>

编译：

更改makefile PREFIX和CC分别设置编译输出路径与编译器。

make && make install，编译完成后相关的bin文件以及lib文件都会在PREFIX指定的目录。

执行sensor可获取监控信息：



通过配置文件可更改显示信息：





lm-sensors的原理就是读取/sys/class/hwmon/hwmon\*目录中的信息，但是注意文件名称，lm-sensors认为temp1\_input就是温度信息，当然1可以是任意数字，但是temp以及input字符都不能变，否则lm-sensors不能解析，其他温度信息或者风扇等其他信息请参照lm-sensors源码的sysfs.c文件。当然如果我们使用devm\_hwmon\_device\_register\_with\_info方法注册hwmon设备属性，则属性名称肯定能与lm-sensors对上。

# 示例说明

示例在regmap的tmp75驱动基础上进一步更改，无需再注册misc设备，直接注册hwmon设备。

示例中温度属性利用标准方法注册，分辨率因为不属于标准属性，则添加到属性组进行了注册。注册完成后在/sys/class/hwmon/hwmon\*/目录下就会出现resolution和temp1\_input，温度只读，分辨率可读写。

测试程序test\_hwmon.c中完成sysfs\_read\_attr()和sysfs\_write\_attr()函数可以对属性文件进行读写，利用这两个函数完成了对温度的读取和分辨率的设置。