LED handling under Linux

If you're reading this and thinking about keyboard leds, these are handled by the input subsystem and the led class is *not* needed.

In its simplest form, the LED class just allows control of LEDs from userspace. LEDs appear in /sys/class/leds/. The maximum brightness of the LED is defined in max_brightness file. The brightness file will set the brightness of the LED (taking a value 0-max_brightness). Most LEDs don't have hardware brightness support so will just be turned on for non-zero brightness settings.

The class also introduces the optional concept of an LED trigger. A trigger is a kernel based source of led events. Triggers can either be simple or complex. A simple trigger isn't configurable and is designed to slot into existing subsystems with minimal additional code. Examples are the ide-disk, nand-disk and sharpsl-charge triggers. With led triggers disabled, the code optimises away.

Complex triggers whilst available to all LEDs have LED specific parameters and work on a per LED basis. The timer trigger is an example. The timer trigger will periodically change the LED brightness between LED_OFF and the current brightness setting. The "on" and "off" time can be specified via /sys/class/leds/<device>/delay_{on,off} in milliseconds. You can change the brightness value of a LED independently of the timer trigger. However, if you set the brightness value to LED_OFF it will also disable the timer trigger.

You can change triggers in a similar manner to the way an IO scheduler is chosen (via /sys/class/leds/<device>/trigger). Trigger specific parameters can appear in /sys/class/leds/<device> once a given trigger is selected.

Design Philosophy

The underlying design philosophy is simplicity. LEDs are simple devices and the aim is to keep a small amount of code giving as much functionality as possible. Please keep this in mind when suggesting enhancements.

LED Device Naming

Is currently of the form:

"devicename:colour:function"

There have been calls for LED properties such as colour to be exported as individual led class attributes. As a solution which doesn't incur as much overhead, I suggest these become part of the device name. The naming scheme above leaves scope for further attributes should they be needed. If sections of the name don't apply, just leave that section blank.

Hardware accelerated blink of LEDs

Some LEDs can be programmed to blink without any CPU interaction. To support this feature, a LED driver can optionally implement the blink_set() function (see <linux/leds.h>). If implemented, triggers can attempt to use it before falling back to software timers. The blink_set() function should return 0 if the blink setting is supported, or -EINVAL otherwise, which means that LED blinking will be handled by software.

The blink_set() function should choose a user friendly blinking value if it is called with *delay_on==0 && *delay_off==0 parameters. In this case the driver should give back the chosen value through delay_on and delay_off parameters to the leds subsystem.

Setting the brightness to zero with brightness_set() callback function should completely turn off the LED and cancel the previously programmed hardware blinking function, if any.

Known Issues

The LED Trigger core cannot be a module as the simple trigger functions would cause nightmare dependency issues. I see this as a minor issue compared to the benefits the simple trigger functionality brings. The rest of the LED subsystem can be modular.

Future Development

At the moment, a trigger can't be created specifically for a single LED. There are a number of cases where a trigger might only be mappable to a particular LED (ACPI?). The addition of triggers provided by the LED driver should cover this option and be possible to add without breaking the current interface.