wuyuwei45的专栏

Android中LCD背光驱动

分类： Linux驱动 Android系统 2013-06-17 16:28 867人阅读 评论(0) 收藏 举报

目录(?) [-]

其实Android的底层就是Linux，所以其驱动本质就是Linux驱动，但是这些Linux驱动是服务上层Android的，所以需遵循上Android的一些接口规范。所以涉及到的Android驱动都应应密切关注上层传递的接口。本文介绍的LCD背光驱动就是从上层一直往下层展现，但是笔者毕竟不是专注于Android上层，碍于知识不充裕，所以对上层的东西介绍得相对简单。

1.Android的Setting

Android的设置里面管理了Andoird系统的所有设置，其中当然包括了屏幕亮度设置。

Setting的源码目录在：

mydroid/packages/apps/Settings/src/com/android/settings

亮度设置的java源文件在：

mydroid/packages/apps/Settings/src/com/android/settings/BrightnessPreference.java

打开这个文件看到：

```
[java]
01. public class BrightnessPreference extends SeekBarDialogPreference implements
02.         SeekBar.OnSeekBarChangeListener, CheckBox.OnCheckedChangeListener {
03.
04.     private SeekBar mSeekBar;
05.     private CheckBox mCheckBox;
06.
07.     private int mOldBrightness;
08.     private int mOldAutomatic;
09.
10.     private boolean mAutomaticAvailable;
11.
12.     private boolean mRestoredOldState;
13.
14.     // Backlight range is from 0 - 255. Need to make sure that user
15.     // doesn't set the backlight to 0 and get stuck
16.     private int mScreenBrightnessDim =
17. ...
```

Android的最上层已经将背光亮度量化为了[0,255]个等级，并且提示注意不要设置为0，所以在进行最低层的背光驱动编写时，可以合理按这个范围部署背光的亮度。

2.Android的背光JNI层

背光的JNI层源码在：

mydroid/frameworks/base/services/jni/com_android_server_LightsService.cpp

这一层就是调用HAL层的方法，为上一层实现一个设置亮度接口。

3.Android的背光HAL层

Java App和JNI一般是google维护的，所以源码位置相对固定，HAL有产品商开发维护的，所以位置是不固定的，

看产品上喜好，笔者使用的TI OMAP4平台，背光的HAL层代码就在：

mydroid/device/ti/xxx_product/liblights/light.c

先浏览欣赏一下light.c先

[cpp]

```
/*
 * Copyright (C) 2008 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */


#define LOG_TAG "lights"

#include <cutils/log.h>

#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <pthread.h>

#include <sys/ioctl.h>
#include <sys/types.h>

#include <hardware/lights.h>

/******************************************************************************/

static pthread_once_t g_init = PTHREAD_ONCE_INIT;
static pthread_mutex_t g_lock = PTHREAD_MUTEX_INITIALIZER;

char const*const LCD_FILE
        = "/sys/class/leds/lcd-backlight/brightness";
char const*const KEYBOARD_FILE
        = "/sys/class/leds/keyboard-backlight/brightness";

char const*const CHARGING_LED_FILE
        = "/sys/class/leds/battery-led/brightness";

/*RGB file descriptors */
char const*const RED_LED_FILE
        = "/sys/class/leds/red/brightness";
char const*const RED_DELAY_ON_FILE
        = "/sys/class/leds/red/delay_on";
char const*const RED_DELAY_OFF_FILE
        = "/sys/class/leds/red/delay_off";
char const*const GREEN_LED_FILE
        = "/sys/class/leds/green/brightness";
char const*const GREEN_DELAY_ON_FILE
        = "/sys/class/leds/green/delay_on";
char const*const GREEN_DELAY_OFF_FILE
        = "/sys/class/leds/green/delay_off";
char const*const BLUE_LED_FILE
        = "/sys/class/leds/blue/brightness";
char const*const BLUE_DELAY_ON_FILE
        = "/sys/class/leds/blue/delay_on";
char const*const BLUE_DELAY_OFF_FILE
        = "/sys/class/leds/blue/delay_off";
...
```

[cpp]

```
static int
set_light_backlight(struct light_device_t* dev,
        struct light_state_t const* state)
```

没碰到过。...

```
{
    int err = 0;
    int brightness = rgb_to_brightness(state);

    pthread_mutex_lock(&g_lock);
    err = write_int(LCD_FILE, brightness);
    pthread_mutex_unlock(&g_lock);

    return err;
}
```

这里关注一下LCD背光的sys文件节点："/sys/class/leds/lcd-backlight/brightness"

疑问1：这个sys接口是谁约定的呢？

疑问2：可不可以改这个接口呢？

疑问3：为什么这里还有其他led的很多sys文件节点呢：

先解释疑问3，因为一个Android设备，不只是有LCD背光的LED，还有可以有其他很多LED（当然也可以没有），所以这里一并实现了这些LED的HAL，产品商可以沿用这些HAL。

先来试验一把：

启动Android设备，在Setting里更改亮度，然后在串口命令行中或"adb shell"中运行命令cat /sys/class/leds/lcd-backlight/brightness，会发现Setting的更改后可以通过cat 显示Setting的更改。所以更加确认了这个"/sys/class/leds/lcd-backlight/brightness"接口是正确的。

因为HAL层是和Linux Kernel交互的，所以这里如果能cat实现读取亮度等级，那么在Linux kernel层就一定实现了sysfs接口。

4.Linux的背光内核层

背光的内核层源码在：driver/leds/led-class.c

这一层由内核开发者去维护，不用我们操心。看看它的init函数

```
[cpp]

static int __init leds_init(void)
{
    leds_class = class_create(THIS_MODULE, "leds");
    if (IS_ERR(leds_class))
        return PTR_ERR(leds_class);
    leds_class->suspend = led_suspend;
    leds_class->resume = led_resume;
    leds_class->dev_attrs = led_class_attrs;
    return 0;
}
```

好明显，正是由于它创建了sys class，名字为"leds"，所以上面的背光sys文件节点"/sys/class/leds/"就有了来由，那么剩下的"../lcd-backlight/brightness"又是怎么来的呢？看看一个注册led设备类的函数.

```
[cpp]

/**
 * led_classdev_register - register a new object of led_classdev class.
 * @parent: The device to register.
 * @led_cdev: the led_classdev structure for this device.
 */
int led_classdev_register(struct device *parent, struct led_classdev *led_cdev)
{
    led_cdev->dev = device_create(leds_class, parent, 0, led_cdev,
                      "%s", led_cdev->name);
    if (IS_ERR(led_cdev->dev))
        return PTR_ERR(led_cdev->dev);

#ifdef CONFIG_LEDS_TRIGGERS
    init_rwsem(&led_cdev->trigger_lock);
...
```

这个注册函数的接口最终会被我们要开发的背光驱动调用，这个接口在/sys/class/leds/下又创建了一个设备接口，名字是led_cdev->name。好明显这里的led_cdev->name应该就是"lcd-backlight"，究竟是不是真的这样呢？继续看。

这个led-class.c在实现两个设备属性，看代码：

```
[cpp]

static struct device_attribute led_class_attrs[] = {
    __ATTR(brightness, 0644, led_brightness_show, led_brightness_store),
```

```
    __ATTR(max_brightness, 0444, led_max_brightness_show, NULL),
#ifdef CONFIG_LEDS_TRIGGERS
    __ATTR(trigger, 0644, led_trigger_show, led_trigger_store),
#endif
    __ATTR_NULL,
};
```

看到了属性名字为”brightness“，这似乎越来越接近解释"/sys/class/leds/lcd-backlight/brightness"的由来了。

5.Linux的背光驱动层

Linux的背光驱动层就是完全由开发者去实现了，其实这层很简单，无非就是通过pwm实现设置背光。在这一层要注意将背光亮度量化为0~255，这是Android上层约定的。

这一层的采用Linux的platform device/driver 模型。它最终会调用Linux内核层leds-class.c的接口。

看看它的probe函数片段：

```cpp
struct display_led_data {
    struct led_classdev pri_display_class_dev;
    struct led_classdev sec_display_class_dev;
    struct omap4_disp_led_platform_data *led_pdata;
    struct mutex pri_disp_lock;
    struct mutex sec_disp_lock;
};
static int omap4_XXX_display_probe(struct platform_device *pdev)
{
    int ret;
    struct display_led_data *info;

    pr_info("%s:Enter\n", __func__);

    if (pdev->dev.platform_data == NULL) {
        pr_err("%s: platform data required\n", __func__);
        return -ENODEV;
    }

    info = kzalloc(sizeof(struct display_led_data), GFP_KERNEL);
    if (info == NULL) {
        ret = -ENOMEM;
        return ret;
    }

    info->led_pdata = pdev->dev.platform_data;
    platform_set_drvdata(pdev, info);

    info->pri_display_class_dev.name = "lcd-backlight";

    info->pri_display_class_dev.brightness_set = omap4_xxx_primary_disp_store;
...


    ret = led_classdev_register(&pdev->dev,
                    &info->pri_display_class_dev);
    if (ret < 0) {
        pr_err("%s: Register led class failed\n", __func__);
        kfree(info);
        return ret;
    }

    if (info->led_pdata->flags & LEDS_CTRL_AS_TWO_DISPLAYS) {
        pr_info("%s: Configuring the secondary LED\n", __func__);
        info->sec_display_class_dev.name = "lcd-backlight2";
        info->sec_display_class_dev.brightness_set =
            omap4_mirage_secondary_disp_store;
        info->sec_display_class_dev.max_brightness = LED_OFF;
        mutex_init(&info->sec_disp_lock);

        ret = led_classdev_register(&pdev->dev,
                        &info->sec_display_class_dev);
...
```

需关注这里设置了info->pri_display_class_dev.name = "lcd-backlight";，然后调用led_classdev_register函数注册，所以完美解释了

led_cdev->name就是 "lcd-backlight"，完美解释了"/sys/class/leds/lcd-backlight/brightness"的由来。

另外info->pri_display_class_dev.brightness_set = omap4_xxx_primary_disp_store;中的omap4_xxx_primary_disp_store()就是我们需要实现的设置背光亮度函数，函数里面就是pwm操作。

到这里又有一个疑问4：为什么只实现设置背光亮度的接口，而没有实现读取当前亮度量化值的接口？

其实Android上层自行处理了这个获取亮度量化值的事情，也就是说Android上层设置了亮度是多少，上层会执行保留设置结果。无需再通过下层读取。

6.总结

6.1针对之前的疑问1/2，其实LCD背光的sys接口路径是可以改的，但是需要Linux内核层和Android HAL层配合来改，单单改一方都是会导致Android Setting无法调节背光。

6.2需注意在实现Linux背光驱动时的亮度量化关系，也就是注意上层传递下来的亮度设置范围是0~255。

6.3Android底层的Linux驱动都是服务于上层Java的，在做Android底层的Linux驱动时需要明确和上层的接口依赖关系，否则无法重用google或芯片厂商实现的接口，从而导致功能无法用。

6.4.再一次证明了Android系统服务运行效率极低。设置背光，其实最终就是设置了一下pwm寄存器即可，但是从Android最顶层一层层调用下来，真是“费尽周折”，怪不得Android设备的硬件配置明显远优于ios设备，但是流畅体验性却不明显优于ios设备。

上一篇 Android/Linux USB Gadget：续

下一篇 GPS调试-1

Angular Cloud Foundry Redis Scala Django Bootstrap