

CSDN新首页上线啦，邀请你来立即体验！(http://feed.csdn.net/)

立即体
验



博客 (http://feed.csdn.net/?ref=toolbar)

学院 (http://edu.csdn.net?ref=toolbar)

下载 (http://download.csdn.net?ref=toolbar)

更多 ▾



0



登录 (https://passport.csdn.net/account/login?ref=toolbar) 注册 (http://passport.csdn.net/account/mobileregister?ref=toolbar&action=mobileRegister)

linux驱动学习2（kpd驱动初步分析）

原创

2013年11月18日 15:37:54

标签：mtk按键驱动 (http://so.csdn.net/so/search/s.do?q=mtk按键驱动&t=blog) /

初步分析 (http://so.csdn.net/so/search/s.do?q=初步分析&t=blog)

2577



shop_ping (http://blog.cs...

+ 关注

(http://blog.csdn.net/King_BingGe)

码云

原创

粉丝

喜欢

未开通

132

129

0

(https://gite
utm_sourc

一、kpd_pdrv_probe函数的分析：

```
[html]
1.  /*1. 输入设备实例  kpd_input_dev */
2.  全局变量：static struct input_dev *kpd_input_dev;
3.
4.  static int kpd_pdrv_probe(struct platform_device *pdev)
5.  {
6.      int i, r;
7.      u16 new_state[KPD_NUM_MEMS];
8.      /* initialize and register input device (/dev/input/eventX) */
9.
10.
11.
12.  /*2. 初始化输入设备并分配内存空间*/
13.      kpd_input_dev = input_allocate_device();
14.      if (!kpd_input_dev)
15.          return -ENOMEM;
16.      /*下面开始填充kpd_input_dev 设备驱动结构体*/
17.      kpd_input_dev->name = KPD_NAME;
18.      kpd_input_dev->id.bustype = BUS_HOST;
19.      kpd_input_dev->id.vendor = 0x2454;
```

他的最新文章

更多文章 (http://blog.csdn.net/King_BingGe)

密集负载下的网卡中断负载均衡smp
affinity及单队列RPS
(/jxnu_xiaobing/article/details/52682291)

/etc/passwd, /etc/shadow
(/jxnu_xiaobing/article/details/48047115)

hadoop2.7完全分布式集群搭建以及任
务测试
(/jxnu_xiaobing/article/details/46931693)



0



```

20.     kpd_input_dev->id.product = 0x6575;
21.     kpd_input_dev->id.version = 0x0010;
22.     kpd_input_dev->open = kpd_open;
23.
24.
25.
26.     /*3. 设置某位为1, 以第二个参数为起始地址, EV_KEY表示要设置的位
27.     作用: 告诉input子系统支持那些事件, EV_KEY 这里表示告诉input子系统支持
28.     按键事件
29.
30.     */
31.     __set_bit(EV_KEY, kpd_input_dev->evbit);
32.
33.
34. #if (KPD_PWRKEY_USE_EINT|KPD_PWRKEY_USE_PMIC)
35.
36.
37.     /*4. 设置某位为1, 以第二个参数为起始地址, EV_KEY表示要设置的位
38.     作用: 告诉input子系统支持那些按键, KPD_PWRKEY_MAP 这里表示告诉input子系统支持
39.     电源按键
40.     */
41.     __set_bit(KPD_PWRKEY_MAP, kpd_input_dev->keybit);
42.     kpd_keymap[8] = 0;
43. #endif
44.     for (i = 17; i < KPD_NUM_KEYS; i += 9) /* only [8] works for Power key */
45.         kpd_keymap[i] = 0;
46.
47.
48.     for (i = 0; i < KPD_NUM_KEYS; i++) {
49.         if (kpd_keymap[i] != 0)
50.             __set_bit(kpd_keymap[i], kpd_input_dev->keybit);
51.     }
52.     /*5. 上述几行代码表示设置电源按键 kpd_keymap 为0, 其它按键 kpd_keymap 为1*/
53.
54.     __set_bit(250, kpd_input_dev->keybit);
55.     __set_bit(251, kpd_input_dev->keybit);
56.
57.
58. #if KPD_AUTOTEST
59.     for (i = 0; i < ARRAY_SIZE(kpd_auto_keymap); i++)
60.         __set_bit(kpd_auto_keymap[i], kpd_input_dev->keybit);
61. #endif
62.
63.
64. #if KPD_HAS_SLIDE_QWERTY
65.     __set_bit(EV_SW, kpd_input_dev->evbit);
66.     __set_bit(SW_LID, kpd_input_dev->swbit);

```



工业键盘



博主专栏



从零开始学OK6410

(<http://blog.csdn.net/column/details/ok6410--study.html>)

27

(<http://blog.csdn.net/column/details/ok6410--study.html>)

--



一步一步学习C++

(<http://blog.csdn.net/column/details/c-plus-plus-.html>)

11

(<http://blog.csdn.net/column/details/c-plus-plus-.html>)

plus-



从零开始学习stm32

(<http://blog.csdn.net/column/details/stm32--study.html>)

26

(<http://blog.csdn.net/column/details/stm32--study.html>)

--

study.html)

在线课程

```

67.     __set_bit(SW_LID, kpd_input_dev->sw);    /* 1: lid shut => closed */
68. #endif
69.
70.
71. #ifdef KPD_PMIC_RSTKEY_MAP
72.     __set_bit(KPD_PMIC_RSTKEY_MAP, kpd_input_dev->keybit);
73. #endif
74.
75.
76.     /*6. 指定kpd_input_dev这个平台设备sysfs中的父设备节点*/
77.     kpd_input_dev->dev.parent = &pdev->dev;
78.     /*7. 注册input输入子系统*/
79.     r = input_register_device(kpd_input_dev);
80.     if (r) {
81.         printk(KPD_SAY "register input device failed (%d)\n", r);
82.         input_free_device(kpd_input_dev);
83.         return r;
84.     }
85.
86.
87.     /* register device (/dev/mt6575-kpd) */
88.     /*7. 指定kpd_dev这个平台设备sysfs中的父设备节点*/
89.     kpd_dev.parent = &pdev->dev;
90.     /*8. 注册混杂设备*/
91.     r = misc_register(&kpd_dev);
92.     if (r) {
93.         printk(KPD_SAY "register device failed (%d)\n", r);
94.         input_unregister_device(kpd_input_dev);
95.         return r;
96.     }
97.
98.     /*8. 注册按键中断*/
99.     /* register IRQ and EINT */
100.    /*9. 设置消抖时间*/
101.    kpd_set_debounce(KPD_KEY_DEBOUNCE);
102.    /*10. 设置中断触发方式*/
103.    mt65xx_irq_set_sens(MT6575_KP_IRQ_ID, MT65xx_EDGE_SENSITIVE);
104.    /*11 . 设置中断优先级*/
105.    mt65xx_irq_set_polarity(MT6575_KP_IRQ_ID, MT65xx_POLARITY_LOW);
106.    /*12. 注册中断处理函数*/
107.    r = request_irq(MT6575_KP_IRQ_ID, kpd_irq_handler, 0, KPD_NAME, NULL);
108.    if (r) {
109.        printk(KPD_SAY "register IRQ failed (%d)\n", r);
110.        misc_deregister(&kpd_dev);
111.        input_unregister_device(kpd_input_dev);
112.        return r;
113.    }

```



0



MY/M在美国点评酒旅移

动端的最佳实践

(utm_source=blog9)

(http://blog.csdn.net/huiyi

Course/detail/603?

log9)



C语言大型软件设计的面向对象

(utm_source=blog9)

(http://blog.csdn.net/huiyi

Course/detail/594?

utm_source=blog9)

热门文章

100条经典C语言笔试题目 (/jxnu_xiaobing/article/details/12561141)

📖 11976

python+pyspider+phantomjs实现简易爬虫功能 (/jxnu_xiaobing/article/details/44983757)

📖 8294

基于stm32f103zet6之使用FSMC驱动TFT的学习 (/jxnu_xiaobing/article/details/8719897)

📖 6655

(转载) STM32-FSMC-LCD详解 (/jxnu_xiaobing/article/details/8718566)

📖 6327

pyspider爬虫的一个应用 (/jxnu_xiaobing/article/details/44671653)

📖 5479

```

114.      /*13. 以下为电源键中断函数的注册*/
115.      #if KPD_PWRKEY_USE_EINT
116.          mt65xx_eint_set_sens(KPD_PWRKEY_EINT, KPD_PWRKEY_SENSITIVE);
117.          mt65xx_eint_set_hw_debounce(KPD_PWRKEY_EINT, KPD_PWRKEY_DEBOUNCE);
118.          mt65xx_eint_registration(KPD_PWRKEY_EINT, true, KPD_PWRKEY_POLARITY,
119.                                  kpd_pwrkey_eint_handler, false);
120.      #endif
121.
122.
123.          if(kpd_enable_lprst && get_boot_mode() == NORMAL_BOOT) {
124.              kpd_print("Normal Boot\n");
125.      #ifdef KPD_PMIC_LPRST_TD
126.              kpd_print("Enable LPRST\n");
127.              /*14. 以下为设置按键唤醒的时间*/
128.              upmu_testmode_pwrkey_rst_en(0x01);
129.              upmu_testmode_homekey_rst_en(0x01);
130.              upmu_testmode_pwrkey_rst_td(KPD_PMIC_LPRST_TD);
131.      #endif
132.          } else {
133.              kpd_print("Disable LPRST %d\n", kpd_enable_lprst);
134.          }
135.          /*15. 设置一个高精度定时器, 并且定义了时间到期的回调函数 aee_timer_func*/
136.          hrtimer_init(&aee_timer, CLOCK_MONOTONIC, HRTIMER_MODE_REL);
137.          aee_timer.function = aee_timer_func;
138.
139.
140.          /*以下为三个按键的初始化, 也就是配置
141.          注意, 默认值gpio输出是0*/
142.      #if 1 // ylliu add. dct default value does not work...
143.          /* KCOL0: GPIO103: KCOL1: GPIO108, KCOL2: GPIO105, KCOL4: GPIO102 input + pull enable + pul
144.          mt_set_gpio_mode(GPIO_KPD_KCOL0_PIN, GPIO_KPD_KCOL0_PIN_M_KP_COL);
145.          mt_set_gpio_dir(GPIO_KPD_KCOL0_PIN, GPIO_DIR_IN);
146.          mt_set_gpio_pull_enable(GPIO_KPD_KCOL0_PIN, GPIO_PULL_ENABLE);
147.          mt_set_gpio_pull_select(GPIO_KPD_KCOL0_PIN, GPIO_PULL_UP);
148.
149.
150.
151.          mt_set_gpio_mode(GPIO_KPD_KCOL1_PIN, GPIO_KPD_KCOL1_PIN_M_KP_COL);
152.          mt_set_gpio_dir(GPIO_KPD_KCOL1_PIN, GPIO_DIR_IN);
153.          mt_set_gpio_pull_enable(GPIO_KPD_KCOL1_PIN, GPIO_PULL_ENABLE);
154.          mt_set_gpio_pull_select(GPIO_KPD_KCOL1_PIN, GPIO_PULL_UP);
155.
156.          mt_set_gpio_mode(GPIO_KPD_KCOL2_PIN, GPIO_KPD_KCOL2_PIN_M_KP_COL);
157.          mt_set_gpio_dir(GPIO_KPD_KCOL2_PIN, GPIO_DIR_IN);
158.          mt_set_gpio_pull_enable(GPIO_KPD_KCOL2_PIN, GPIO_PULL_ENABLE);
159.          mt_set_gpio_pull_select(GPIO_KPD_KCOL2_PIN, GPIO_PULL_UP);
160.

```



```

161.     mt_set_gpio_mode(GPIO_KPD_KCOL4_PIN, GPIO_KPD_KCOL4_PIN_M_KP_COL);
162.     mt_set_gpio_dir(GPIO_KPD_KCOL4_PIN, GPIO_DIR_IN);
163.     mt_set_gpio_pull_enable(GPIO_KPD_KCOL4_PIN, GPIO_PULL_ENABLE);
164.     mt_set_gpio_pull_select(GPIO_KPD_KCOL4_PIN, GPIO_PULL_UP);
165.
166.
167.     /* KROW0: GPIO98, KROW1: GPIO97: KROW2: GPIO95 output + pull disable + pull down */
168.     mt_set_gpio_mode(GPIO_KPD_KROW0_PIN, GPIO_KPD_KROW0_PIN_M_KP_ROW);
169.     mt_set_gpio_dir(GPIO_KPD_KROW0_PIN, GPIO_DIR_OUT);
170.     mt_set_gpio_pull_enable(GPIO_KPD_KROW0_PIN, GPIO_PULL_DISABLE);
171.     mt_set_gpio_pull_select(GPIO_KPD_KROW0_PIN, GPIO_PULL_DOWN);
172.
173.     // mt_set_gpio_mode(97, 1);
174.     // mt_set_gpio_dir(97, 1);
175.     // mt_set_gpio_pull_enable(97, 0);
176.     // mt_set_gpio_pull_select(97, 0);
177.     //
178.     // mt_set_gpio_mode(95, 1);
179.     // mt_set_gpio_dir(95, 1);
180.     // mt_set_gpio_pull_enable(95, 0);
181.     // mt_set_gpio_pull_select(95, 0);
182. #endif
183.
184.     // default disable backlight. reboot from recovery need this.
185.     kpd_disable_backlight();
186.
187.     // store default state, resolve recovery bugs.
188.     kpd_get_keymap_state(new_state);
189.     memcpy(kpd_keymap_state, new_state, sizeof(new_state));
190.
191.     return 0;
192. }

```

二、当执行完probe函数进行相关初始化后，这时候，当我们按键按下了，就会触发中断，进入中断服务子程序

[html]

```
1. static irqreturn_t __tcmfunc kpd_irq_handler(int irq, void *dev_id)
```

```

2.  {
3.      /* use _nosync to avoid deadlock */
4.      disable_irq_nosync(MT6575_KP_IRQ_ID);
5.      tasklet_schedule(&kpd_keymap_tasklet);
6.      return IRQ_HANDLED;
7.  }

```



0



可以看到，中断服务程序里面执行了 `tasklet_schedule(&kpd_keymap_tasklet);` 跟踪代码可以发现，实际上是执行了这个函数 `kpd_keymap_handler`，下面仔细分析这个函数，详细注释如下：

```

[html]
1.  static void kpd_keymap_handler(unsigned long data)
2.  {
3.      int i, j;
4.      bool pressed;
5.      u16 new_state[KPD_NUM_MEMS], change, mask;
6.      u16 hw_keycode, linux_keycode;
7.      kpd_get_keymap_state(new_state);
      首先读取键值，并且存放于new_state中
8.
9.
10.     if (pmic_get_acc_state() == 1) {
11.         for (i = 0; i < KPD_NUM_MEMS; i++) {
12.             change = new_state[i] ^ kpd_keymap_state[i];
13.             if (!change)
14.                 continue;
15.
16.             for (j = 0; j < 16; j++) {
17.                 mask = 1U << j;
18.                 if (!(change & mask))
19.                     continue;
20.
21.                 hw_keycode = (i << 4) + j;
22.                 //i = 0, j = 1;
23.                 //这里是得到hw_keycode的值
24.                 printk("hw_keycode = %d , i = %d, j = %d \n", hw_keycode, i, j);

```

```

25.         /* bit is 1: not pressed, 0: pressed */
26.         pressed = !(new_state[i] & mask);    //(new_state[i] & mask) = 0
27.         if (kpd_show_hw_keycode) {
28.             printk(KPD_SAY "(%s) HW keycode = %u\n",
29.                 pressed ? "pressed" : "released",
30.                 hw_keycode);
31.         }
32.         BUG_ON(hw_keycode >= KPD_NUM_KEYS);
33.         linux_keycode = kpd_keymap[hw_keycode];
34.         printk("linux_keycode = %d  \n", linux_keycode);
35.
36.         if(unlikely(linux_keycode == 0)) {
37.             if (hw_keycode == 1 && pressed) { // special key, SOS.
38.                 struct device *dev = &(kpd_input_dev->dev);
39.                 char *envp[] = { "SOS_pressed", NULL };
40.                 kobject_uevent_env(&dev->kobj, KOBJ_CHANGE, envp);          //建立设备
41.                 printk(KPD_SAY "SOS_pressed\n");
42.                 // used by recovery.
43.                 /*这个接口会向INPUT子系统上报按键 (该按键被按下)*/
44.                 input_report_key(kpd_input_dev, 251, pressed);          //如果
45.                 } else if (hw_keycode == 2 && pressed) { // special key, background.
46.                     struct device *dev = &(kpd_input_dev->dev);
47.                     char *envp[] = { "background_pressed", NULL };
48.                     kobject_uevent_env(&dev->kobj, KOBJ_CHANGE, envp);
49.                     printk(KPD_SAY "background_pressed\n");
50.                     // used by recovery.
51.                     input_report_key(kpd_input_dev, 8, pressed);
52.                 } else if (hw_keycode == 4 && pressed) { // special key, mode.
53.                     struct device *dev = &(kpd_input_dev->dev);
54.                     char *envp[] = { "mode_pressed", NULL };
55.                     kobject_uevent_env(&dev->kobj, KOBJ_CHANGE, envp);
56.                     printk(KPD_SAY "mode_pressed\n");
57.                 } else if (hw_keycode == 1 || hw_keycode == 2 || hw_keycode == 4) { // add this
58.                     printk(KPD_SAY "background or SOS or mode release!\n");
59.                     // used by recovery.
60.                     if (hw_keycode == 1)
61.                         input_report_key(kpd_input_dev, 251, pressed);
62.                     else if (hw_keycode == 2)
63.                         input_report_key(kpd_input_dev, 8, pressed);
64.                 } else {
65.                     kpd_print("Linux keycode = 0\n");
66.                     continue;
67.                 }
68.             }

```

这里的linux_keycode恒为零。

文件？

上层检测到SOS_pressed就会做相应处理。

```
69.         kpd_aee_handler(linux_keycode, pressed);
70.         kpd_backlight_handler(pressed, linux_keycode);
71.         input_report_key(kpd_input_dev, linux_keycode, pressed);
72.     }
73. }
74. } else {
75.     printk(KPD_SAY "acc off, ignore and key...\n");
76. }
77.
78. memcpy(kpd_keymap_state, new_state, sizeof(new_state));
79.
80.
81. kpd_print("save new keymap state\n");
82. enable_irq(MT6575_KP_IRQ_ID);
83. }
```



0



三、kpd_aee_handler函数分析

```
[html]
1. static void kpd_aee_handler(u32 keycode, u16 pressed) {
2.     if(pressed) {
3.         if(keycode == KEY_VOLUMEUP) {
4.             __set_bit(0, &aee_pressed_keys);
5.         } else if(keycode == KEY_VOLUMEDOWN) {
6.             __set_bit(1, &aee_pressed_keys);
7.         } else {
8.             return;
9.         }
10.        kpd_update_aee_state();
11.    } else {
12.        if(keycode == KEY_VOLUMEUP) {
13.            __clear_bit(0, &aee_pressed_keys);
14.        } else if(keycode == KEY_VOLUMEDOWN) {
15.            __clear_bit(1, &aee_pressed_keys);
16.        } else {
17.            return;
18.        }
19.        kpd_update_aee_state();
20.    }
21. }
```


详细分析：

1.__set_bit(0, &aee_pressed_keys), 定义了一个：static u16 aee_pressed_keys;

所以__set_bit的意思是将aee_pressed_keys的bit0设置为1

2.相应的__clear_bit(0, &aee_pressed_keys);就是把aee_pressed_keys的bit0清零，

3.还有在内核的non-atomic.h文件中还有一些其它的位操作，记住__set_bit和set_bit的区别就是前者是非原子操作，而后者是原子操作，所谓原子操作，意思是最小的执行单位，再其执行过程中是不会被其他任务打断的。

四、背光处理函数

```
[html]
1. void kpd_backlight_handler(bool pressed, u16 linux_keycode)
2. {
3.     if (kpd_suspend && !test_bit(linux_keycode, kpd_wake_keybit)) {
4.         kpd_print("Linux keycode %u is not WAKE key\n", linux_keycode);
5.         return;
6.     }
7.     /* not in suspend or the key pressed is WAKE key */
8.     if (pressed) {
9.         atomic_inc(&kpd_key_pressed);
10.        kpd_backlight_on = !!atomic_read(&kpd_key_pressed);
11.        schedule_work(&kpd_backlight_work);    //点亮背光灯
12.        kpd_print("switch backlight on\n");
13.    } else {
14.        atomic_dec(&kpd_key_pressed);
15.        mod_timer(&kpd_backlight_timer,          //KPD_BACKLIGHT_TIME控制背光时间，单位为sec，
如果注释掉这句，背光将不灭
16.            jiffies + KPD_BACKLIGHT_TIME * HZ);
17.        kpd_print("activate backlight timer\n");
18.    }
19. }
```

详细分析

1.首先用到了一个位操作函数，注意这个函数是原子操作test_bit

2.全局变量static atomic_t kpd_key_pressed = ATOMIC_INIT(0);这是原子操作的初始化，

kpd_key_pressed初始化为0

3.上述函数涉及到一些原子操作函数，解释如下：

atomic_inc(&kpd_key_pressed); 是对变量进行加1操作

atomic_dec(&kpd_key_pressed); 是对变量进行减1操作

!!atomic_read(&kpd_key_pressed);是读取变量的值，前面两个 !!强调该返回值不是1就是0：bool类型

4.mod_timer：该函数的作用是修改一个已经调度的定时器结构的到期时间。



0



五、背光控制函数

调度的是这个函数

```
[html]
1. static void kpd_switch_backlight(struct work_struct *work)
2. {
3.     if (kpd_backlight_on) {
4.         kpd_enable_backlight();
5.         kpd_print("backlight is on\n");
6.     } else {
7.         kpd_disable_backlight();
8.         kpd_print("backlight is off\n");
9.     }
10. }
```

这里就能够看到使能和失能背光的函数，继续跟踪：

```
[html]
1. void kpd_enable_backlight(void)
2. {
3.     /*mt6326_kpled_dim_duty_Full();
4.     mt6326_kpled_Enable();*/
5.     upmu_kpled_dim_duty(31);
6.     upmu_kpled_en(1);
7. }
8. upmu_kpled_dim_duty这是控制背光电流大小从而可以控制亮度
9. upmu_kpled_en这是控制开关。
```

mod_timer函数的补充

http://www.360doc.com/content/12/0510/11/6973384_210041084.shtml

(http://www.360doc.com/content/12/0510/11/6973384_210041084.shtml)



0



kpd驱动初步分析完毕。

版权声明：本文为博主原创文章，未经博主允许不得转载。



相关文章推荐

Android增加一个物理按键检测步骤 (/u010177751/article/details/38520245)

Android增加一个物理按键检测步骤



u010177751 (<http://blog.csdn.net/u010177751>) 2014-08-12 20:11 1135

修改mtk平台power按键的gpio控制口 (/qq_24614807/article/details/71813645)

根据项目需要，现在需要将项目中的power按键原有的控制方式改成GPIO86口。在原有的项目中，GPIO86原本是用来控制矩阵键盘中的一行，现在需要将dws文件进行相应的修改，需要将GPIO86的初...



qq_24614807 (http://blog.csdn.net/qq_24614807) 2017-05-13 15:11 455

个人开发者如何通过人工智能盈利？



个人如何开发一款人工智能应用？个人如何利用免费的人工智能工具与平台赚钱？

(http://www.baidu.com/cb.php?c=lgF_pyfqHmknjcvPW60IZ0qnfK9ujYzP1fvn1DL0Aw-5Hc4nHb3rjD0TAq15HfLPWRznjb0T1YYnhFbn1RYPHT3uycYPjDv0AwY5HDdnH01P1bkn100lgF_5y9YIZ0IQzqMpgwBUvqoQhP8QvIGIAPCmgfEmyPYpguGIZbEPH-ujN-njT4rH7buWTsuyPbPyDsnyRz5LNYUNq1ULNzmvRqnHDknAwBUAqM0ZFb5HD0mhYqn0KsTWYs0ZNGujYkPHTYn1mk0AqGujYkn10snjf10APGujYLnWm4n1c0ULI85H00TZbqnW0)



android6.0按键处理浅析 (/liyanfei123456/article/details/53196693)

处理流程及示意图：1.硬件配置：kernel-3.18\arch\arm\boot\dtbs\projectxxx.dts & keypad { ...



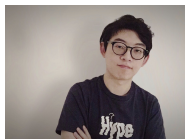
liyanfei123456 (<http://blog.csdn.net/liyanfei123456>) 2016-11-17 10:31 645

在MTK6572当中如何增加一个新按键，而且此按键值在虚拟按键当中可以使用 (/u013478557/a...

在物理按键当中如果你要增加一个新按键：[Description] How to add a new key on android ICS/ICS2 [Solution] 1.在DCT to ol ...



u013478557 (<http://blog.csdn.net/u013478557>) 2014-01-14 17:47 2336




Webpack 前端工程化入门

如果你想了解在公司级别的大型项目中是如何做工程化的？如果你想深入了解 Webpack，并且想用它的各种特性来提升构建效率？那么你就有理由来了解一下这门课。

(http://www.baidu.com/cb.php?c=lgF_pyfqHmknjcvPjR0IZ0qnfK9ujYzP1ndnHfY0Aw-5Hc4nj6vPjm0TAq15Hf4rjn1n1b0T1YzuHFBAPWuHbdPj04n1cd0AwY5HDdnH01P1bkn100lgF_5y9YIZ0IQzqMpgwBUvqoQhP8QvIGIAPCmgfEmvq_lyd8Q1R4uH0vPymvPAmLuhF-PHRduHfLrHcsPjqdIAdxTvqdThP-5HDknHK3mhkEusKzujYk0AFV5H00TZcqN0KdpYfqHRLPjnvnfKEpyfqHnsnj0YnsKWpyfqP1cvrHnz0AqLUWYs0ZK45HcsP6KWThnqnHcLn6)

MTK LIGHT 代码分析 (/chiooo/article/details/44021057)

MTK LIGHT 代码分析项目上需要做些定制化的东西，需要用到light 一块的东西，好久以前看过，但是没有记录下来，这次重新看看，然后记录下来。lightservice startprivate ...

 chiooo (<http://blog.csdn.net/chiooo>) 2015-03-02 17:47 1284




linux驱动内核学习下学期2 (<http://download.csdn.net/detail/zjshaoxingjj/8...>)

[/http://download.csdn.net/detail/zjshaoxingjj/8...](http://download.csdn.net/detail/zjshaoxingjj/8...) 2015-07-09 22:35 7.70MB [下载](#)


linux驱动—input输入子系统—The simplest example（一个最简单的实例）分析（2）(/tianx...

linux驱动—input输入子系统—The simplest example（一个最简单的实例）分析（1）的地址链接 1、上一篇说到了input_match_device函数，这篇接着...

 tianxiawuzhei (<http://blog.csdn.net/tianxiawuzhei>) 2012-05-27 22:23 2782

Linux驱动I2C分析 (/tanxjian/article/details/7328353)

一：前言 I2c是philips提出的外设总线。I2C只有两条线，一条串行数据线：SDA,一条是时钟线SCL.正因为这样，它方便了工程人员的布线。另外，I2C是一种多主机控制总线。它和USB总线...

 tanxjian (<http://blog.csdn.net/tanxjian>) 2012-03-07 13:14 1714



Linux驱动入门学习(三)：I2C架构全面理解 (/sdsh1880gm/article/details/53559037)

I2C 概述 I2C是philips提出的外设总线。 I2C只有两条线,一条串行数据线:SDA,一条是时钟线SCL，使用SCL，SDA这两根信号线就实现了设备之间的数据交互，它方...

 sdsh1880gm (<http://blog.csdn.net/sdsh1880gm>) 2016-12-10 16:00 115



Linux驱动学习--时间、延迟及延缓操作2 (/dahailinan/article/details/6869289)

延迟执行 设备驱动常常需要延后一段时间执行一个特定片段的代码,常常允许硬件完成某个任务.长延迟 有时，驱动需要延后执行相对长时间，长于一个时钟嘀嗒。忙等待(尽量别用) 若想延迟执行若干...

 dahailinan (<http://blog.csdn.net/dahailinan>) 2011-10-13 11:15  418



Linux驱动学习--时间、延迟及延缓操作2 (/cat_lover/article/details/6969639)

Linux驱动学习--时间、延迟及延缓操作2 2008-05-24 17:55 延迟执行 设备驱动常常需要延后一段时间执行一个特定片段的代码, 常常允许硬件完成某个任务. ...

 cat_lover (http://blog.csdn.net/cat_lover) 2011-11-14 18:03  500

linux驱动学习第二天 (linux内核及其编程2) (/czh52911/article/details/7260539)

一、linux内核的编译及加载 编译内核需要先配置内核, 使用命令 #make menuconfig ...

 czh52911 (<http://blog.csdn.net/czh52911>) 2012-02-15 12:57  564

linux驱动学习之内核线程分析 (/fontlose/article/details/8291674)

内核线程和普通的进程间的区别在于内核线程没有独立的地址空间,它只在 内核空间运行, 从来不切换到用户空间去; 并且和普通进程一样, 可以被调度, 也可以被抢占。 一 线程的创建 s...

 fontlose (<http://blog.csdn.net/fontlose>) 2012-12-13 17:29  9535


linux驱动学习之内核线程分析 (/wilsonbolui/article/details/22265389)

内核线程和普通的进程间的区别在于内核线程没有独立的地址空间,它只在 内核空间运行, 从来不切换到用户空间去; 并且和普通进程一样, 可以被调度, 也可以被抢占。 一 线程的创建 [p...

 Wilsonbolui (<http://blog.csdn.net/Wilsonbolui>) 2014-03-27 09:17  5457

linux驱动学习之tasklet分析 (/fontlose/article/details/8279113)

tasklet是中断处理下半部分最常用的一种方法, 驱动程序一般先申请中断, 在中断处理函数内完成中断上半部分的工作后调用tasklet。 tasklet有如下特点: 1.tasklet只可以在一个C...

 fontlose (<http://blog.csdn.net/fontlose>) 2012-12-10 17:20 5348



Linux驱动文件包2 (<http://download.csdn.net/detail/hxnai/1935288>)

[/http://download.csdn.net/detail/hxnai/1935288](http://download.csdn.net/detail/hxnai/1935288) 2009-12-25 21:54 13.42MB

下载




Linux驱动入门--LDD2_CN.pdf (<http://download.csdn.net/detail/lzc336/15...>)

[/http://download.csdn.net/detail/lzc336/15...](http://download.csdn.net/detail/lzc336/15...) 2009-07-29 10:33 2.20MB

下载

arm-linux驱动：初步字符设备 ([u010650281/article/details/51437951](http://blog.csdn.net/u010650281/article/details/51437951))

linux驱动大致分成三个种类，1,字符设备，，，最简单最常用的一种，2,块设备，，，比较麻烦，但是大都功能很强大，3,网络设备，，，这个就不说了，最麻烦的一类，但是也是最单一，技术最成...

 u010650281 (<http://blog.csdn.net/u010650281>) 2016-05-17 17:37 182




嵌入式Linux驱动程序设计从入门到精通：ffe0844001a2f011b.zip ([http://do...](http://download.csdn.net/detail/ffe0844001a2f011b/1111111))

[/http://download.csdn.net/detail/ffe0844001a2f011b/1111111](http://download.csdn.net/detail/ffe0844001a2f011b/1111111) 2016-10-11 19:20 81.43MB

下载

第一个linux驱动_读写设备文件（2）([ddway12/article/details/53438804](http://blog.csdn.net/ddway12/article/details/53438804))

-----在android模拟器和开发板上进行测试驱动模块 第一步添加逻辑功能部分代码 1.添加建立文件部分代码，添加后的hello.c的代码如下：`#include #include #...`

 ddway12 (<http://blog.csdn.net/ddway12>) 2016-12-02 20:17 684