实验 10 嵌入式 LINUX 下 LED 驱动程序

实验目的

- 掌握简单的字符设备驱动程序的编写、编译、测试方法。

实验设备

- 硬件：PC 机+Tiny6410 开发板
- 软件：WINXP+VMWARE 虚拟机+UBUNTU10.04-32bit+arm-linux-gcc

实验内容

- 1.编写 LED 驱动程序 led_drv.c，用来同时控制 led1~led4 的亮灭
- 2.编译驱动程序，装载到内核，
- 3.编写测试程序，观察 LED1~LED4 的亮灭情况。观看运行结果。

实验原理

参考第 7 章相关部分

实验步骤

1) 在虚拟 UNBUNT 下编辑 LED 驱动源程序 led_drv.c

2)编辑驱动程序的 Makefile 文件，执行 make，生成驱动模块文件 led_drv.ko。注意：在编译驱动之前，应先把开发板上运行的 LINUX 内核源码安装在如下目录/opt/FriendlyARM/mini6410/linux/，并确保已经对内核至少编译了一次（make zImage）

3) 在虚拟 UNBUNT 下编辑 LED 驱动测试源程序 led_drv_test.c

4)执行如下命令把测试程序编译成可执行程序 leddrvtest：

arm-linux-gcc -o leddrvtest led_drv_test.c

5) 把驱动模块 led_drv.ko 和测试程序 leddrvtest 拷贝到 windows 下。在虚拟 LINUX 这个文件所在目录执行如下命令：

cp leddrvtest led_drv.ko /mnt/hgfs/share

指令执行成功后，会在 WINDOWS 的“D:\VM_OS\ub100432bit\share”目录下看到这两个文件。

4) 打开超级终端（超级终端配置参考“Tiny6410 设置超级终端.pdf”）。使用串口线把开发板的 COM0 口和 PC 的串口连接起来，把开发板的 S2 开关拨到 NAND 一侧，连接电源线，然后上电启动开发板，此时开发板上的 LINUX 系统开始启动，会在超级终端看到启动信息。

5) 等到开发板的 LINUX 系统启动完成后，使用超级终端，通过串口把驱动模块 led_drv.ko 和测试程序 leddrvtest 下载到开发板：超级终端传送菜单->发送文件，打开发送文件对话框，点击文件名后的浏览按钮，找到要发送的文件，协议选择 Zmodem 与崩溃恢

复，点击发送，文件就发送到嵌入式 LINIUX 的当前目录下。注意：如果想重新发送某文件，应先把开发板上原来的文件删掉，然后再发送。新发送的文件不会覆盖开发板上原来的同名文件。

以下步骤所涉及到的操作，均针对开发板上的嵌入式 LINUX 操作系统，因此以下步骤涉及到的命令均在超级终端窗口下达。

6) 在超级终端窗口（此时的超级终端就是开发板上的 LINUX 的默认输出设备，可通过超级终端对开发板上的 LINUX 系统下达指令）执行如下命令，把 LED 驱动模块加载到内核：

insmod led_drv.ko

然后在超级终端窗口执行指令：cat /proc/devices

如果能看到 leddrv 设备，则说明驱动模块加载成功

或者在/dev 目录下看到 myled_drv 文件，也说明驱动模块加载成功

或者用 lsmod 命令查看模块，如果发现有 led_drv 模块，则说明加载成功

若要重新加载该驱动模块，则应先使用 rmmod 命令卸载该驱动模块，然后再加载

rmmod leddrv

7)在超级终端窗口输入如下命令：/etc/rc.d/init.d/leds stop。该命令将停止 led-player 对 led 的操纵（led-player 是嵌入式 LINUX 系统已经嵌入进去的 LED 点亮程序，该程序如不停止，那么用户开发的 LED 程序将不能调用 LED 设备）。

8)为测试程序增加可执行权限 chmod+x leddrvtest，然后执行

./ leddrvtest on，四个 LED 灯全亮

./ leddrvtest off，四个 LED 灯全灭

观察记录 LED 的点亮情况。

9) 修改驱动程序程序，实现如下功能：当应用程序往 LED 设备分别写入 1~4 时，分别点亮 LED1~4，写入 5~8 时，分别熄灭 LED1~LED4，重新编译驱动程序，并编写相应的测试程序进行测试。

10) 体会嵌入式 LINUX 下驱动程序的编写、编译、加载、测试流程。

参考源程序

led_drv.c 源代码：

```
#include <linux/miscdevice.h>
#include <linux/delay.h>
#include <asm/irq.h>
//#include <mach/regs-gpio.h>
#include <mach/hardware.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/mm.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <linux/delay.h>
#include <linux/moduleparam.h>
```

```
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/ioctl.h>
#include <linux/cdev.h>
#include <linux/string.h>
#include <linux/list.h>
#include <linux/pci.h>
#include <asm/uaccess.h>
#include <asm/atomic.h>
#include <asm/unistd.h>
#include <mach/map.h>
#include <mach/regs-clock.h>
#include <mach/regs-gpio.h>
#include <plat/gpio-cfg.h>
#include <mach/gpio-bank-e.h>
#include <mach/gpio-bank-k.h>
#include <linux/device.h>

static struct class *led_drv_class;
//static struct class_device      *first_drv_class_dev;

volatile unsigned long *gpkcon0=NULL;
volatile unsigned long *gpkdat=NULL;

static int led_drv_open(struct inode *inode,
                                                struct file *file)
{
    //printk("led_drv_open\n");
    //配置 GPK4/5/6/7 为输出功能
    *gpkcon0    &=    ~(0xffff<<16);//GPKCON0[31:16]清零
    *gpkcon0 |=    (0x1<<(4*4)) |(0x1<<(5*4))|(0x1<<(6*4))|(0x1<<(7*4));
        //配置 GPK4/5/6/7 为输出功能
    return 0;
}

static ssize_t led_drv_write(struct file *file,
                                                const char __user *buf,
                                                size_t count,
                                                loff_t * ppos)
{
    //printk("led_drv_write\n");
    int val;
    copy_from_user(&val, buf, count);//用户空间到内核空间传递数据
    if(val==1)
```

```
            {
            //LED1~LED4 全亮
            *gpkdat &= ~(0xf<<4);//GPKDAT[0:4]清零
            *gpkdat |= (0x0<<4);//输出低电平，LED 点亮
            }
        else if(val == 0)
            {
            //LED1~LED4 全灭
            *gpkdat &= ~(0xf<<4);//GPKDAT[0:4]清零
            *gpkdat |= (0xf<<4);//输出高电平，LED 熄灭
            }

        return 0;
}

static struct file_operations led_drv_fops = {
            .owner    =     THIS_MODULE,
/* 这是一个宏，推向编译模块时自动创建的__this_module 变量 */
            .open     =     led_drv_open,
        .write     =         led_drv_write,
};

int major;
int led_drv_init(void)
{
        major=register_chrdev(0, "leddrv", &led_drv_fops);

        led_drv_class = class_create(THIS_MODULE, "led_drv");
          device_create(led_drv_class, NULL, MKDEV(major, 0), NULL, "myled_drv");
        //创建设备节点:/dev/myled_drv，主设备号为 major，次设备号为 0
        gpkcon0=(volatile unsigned long *)ioremap(0x7F008800,16);//ioremap 第 1 个参数为起始地
址，第 2 个参数为长度；
        gpkdat=gpkcon0+2;//这里加 2 实际上是加 2 个 long 型变量长度，加 8 个字节



        return 0;
}

void led_drv_exit(void)
{
        unregister_chrdev(major, "leddrv");

        //class_device_unregister(first_drv_class_dev);
        device_destroy(led_drv_class,MKDEV(major, 0));
```

```
	class_destroy(led_drv_class);

	iounmap(gpkcon0);
}

module_init(led_drv_init);
module_exit(led_drv_exit);

MODULE_AUTHOR("YHD");//驱动程序所有者
MODULE_DESCRIPTION("THE LED DRIVER");//描述信息
MODULE_LICENSE("GPL");//遵循的协议
```

led_drv_test.c 源程序：

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

/*
leddrvtest on:打开 LED1~LED4
leddrvtest off:熄灭 LED1~LED4
*/
int main(int argc, char **argv)
{
	int fd;
	int val = 1;

	fd = open("/dev/myled_drv", O_RDWR);

	if (fd < 0)
	{
		printf("can't open!\n");
	}

	if (argc != 2)
		{
			printf("Usage:\n");
			printf("%s <on|off>\n",argv[0]);
			return 0;
		}

	if (strcmp(argv[1], "on") == 0)
```

```
        {
            val = 1;
        }
    else
        {
            val = 0;
        }

    write(fd, &val, 4);

    return 0;
}
```

Makefile：

```
obj-m += led_drv.o
KDIR := /opt/FriendlyARM/mini6410/linux/linux-2.6.38 #嵌入式 LINUX 内核源码路径

PWD = $(shell pwd)
all:
    make -C $(KDIR) M=$(PWD) modules
clean:
    rm -rf *.o
```