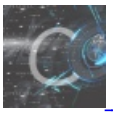
Linux驱动开发|MISC驱动

原创

安迪西 于 2021-12-28 16:14:21 发布

分类专栏： Linux驱动开发 文章标签： 驱动开发 linux 运维 misc platform



MISC驱动

一、MISC设备驱动介绍

MISC设备也叫杂项设备，即当板子上的某些外设无法分类时，就可以使用MISC设备驱动。所有的 MISC 设备驱动的主设备号都为 10，不同的设备使用不同的从设备号。MISC设备会自动创建cdev，不需要手动创建，因此采用 MISC 设备驱动可以简化字符设备驱动的编写。

向 Linux 注册一个 miscdevice 设备， miscdevice是一个结构体，定义在文件 include/linux/miscdevice.h 中，内容如下：

```
struct miscdevice {
int minor;        /* 子设备号 */
const char *name;       /* 设备名字 */
const struct file_operations *fops; /* 设备操作集 */
struct list_head list;
struct device *parent;
struct device *this_device;
const struct attribute_group **groups;
const char *nodename;
umode_t mode;
};
```

定义了MISC设备后，需要设置minor、name和fops三个成员变量。minor表示子设备号，主设备号固定为10，Linux系统预定义了一些MISC设备的子设备号，定义在include/linux/miscdevice.h 文件中，可以从中挑选，也可以自己定义

```
#define PSMOUSE_MINOR 1
#define MS_BUSMOUSE_MINOR 2 /* unused */
#define ATIXL_BUSMOUSE_MINOR 3 /* unused */
/*#define AMIGAMOUSE_MINOR 4 FIXME OBSOLETE */
#define ATARIMOUSE_MINOR 5 /* unused */
#define SUN_MOUSE_MINOR 6 /* unused */
......
#define MISC_DYNAMIC_MINOR 255
```

设置好miscdevice后，就需要向系统中注册MISC设备，卸载设备驱动时需要注销掉MISC设备。注册和注销MISC设备使用如下函数

- misc_register：注册MISC设备

```
int misc_register(struct miscdevice * misc)
//misc：要注册的 MISC 设备
//返回值：负数，失败；0，成功
/*************************************************************/
/*******该函数会自动创建设备，包含了传统设备驱动中的下列步骤*******/
/*************************************************************/
alloc_chrdev_region();  /* 申请设备号 */
cdev_init();    /* 初始化 cdev */
cdev_add();    /* 添加 cdev */
class_create();   /* 创建类 */
device_create();   /* 创建设备 */
```

- misc_deregister：注销MISC设备

```
int misc_deregister(struct miscdevice * misc)
//misc：要注销的 MISC 设备
//返回值：负数，失败；0，成功
/*************************************************************/
/*******该函数会自动删除设备，包含了传统设备驱动中的下列步骤*******/
/*************************************************************/
cdev_del();      /* 删除 cdev */
unregister_chrdev_region();  /* 注销设备号 */
device_destroy();     /* 删除设备 */
class_destroy();     /* 删除类 */
```

二、MISC设备驱动实验

本实验采用 platform 加 misc 的方式编写 beep 驱动，采用 platform 来实现总线、设备和驱动， misc 主要负责完成字符设备的创建

2.1 修改设备树

- 添加pinctrl节点：在iomuxc节点的imx6ul-evk子节点下创建“pinctrl_beep”节点，复用SNVS_TAMPER1

```
pinctrl_beep: beepgrp {
 fsl,pins = <
  MX6ULL_PAD_SNVS_TAMPER1__GPIO5_IO01  0x10B0
 >;
};
//MX6ULL_PAD_SNVS_TAMPER1__GPIO5_IO01 用于设置pin的复用功能
//0x10B0 用于设置pin的电气特性
```

- 添加BEEP设备节点：在根节点下创建beep设备节点，设置PIN对应的pinctrl节点，指定所使用的的GPIO

```
beep {
 #address-cells = <1>;
 #size-cells = <1>;
 compatible = "atkalpha-beep";
 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_beep>;
 led-gpio = <&gpio5 1 GPIO_ACTIVE_HIGH>;
 status = "okay";
};
```

- 检查PIN是否冲突：检查pinctrl中设置以及设备节点中指定的引脚有没有被别的外设使用

保存修改后，在kernel主目录下使用“make dtbs”命令编译设备树，使用新的设备树文件启动Llinux系统

2.2 驱动程序编写

新建 miscbeep.c 驱动文件，并输入如下内容

```
#define MISCBEEP_NAME  "miscbeep" /* 名字  */
#define MISCBEEP_MINOR  144   /* 子设备号 */
#define BEEPOFF    0   /* 关蜂鸣器 */
#define BEEPON     1   /* 开蜂鸣器 */
/* miscbeep设备结构体 */
struct miscbeep_dev{
 dev_t devid;    /* 设备号   */
 struct cdev cdev;   /* cdev  */
 struct class *class;  /* 类   */
 struct device *device;  /* 设备   */
 struct device_node *nd;  /* 设备节点 */
 int beep_gpio;    /* beep所使用的GPIO编号  */
};
struct miscbeep_dev miscbeep;  /* beep设备 */
/* 打开设备 */
static int miscbeep_open(struct inode *inode, struct file *filp){
 filp->private_data = &miscbeep; /* 设置私有数据 */
 return 0;
}
/* 向设备写数据 */
static ssize_t miscbeep_write(struct file *filp, const char __user *buf, size_t cnt, loff_t *offt){
 int retvalue;
 unsigned char databuf[1];
 unsigned char beepstat;
 struct miscbeep_dev *dev = filp->private_data;

 retvalue = copy_from_user(databuf, buf, cnt);
 if(retvalue < 0) {
  printk("kernel write failed!\r\n");
  return -EFAULT;
 }

 beepstat = databuf[0];  /* 获取状态值 */
 if(beepstat == BEEPON) {
  gpio_set_value(dev->beep_gpio, 0); /* 打开蜂鸣器 */
 } else if(beepstat == BEEPOFF) {
  gpio_set_value(dev->beep_gpio, 1); /* 关闭蜂鸣器 */
 }
 return 0;
}
/* 设备操作函数 */
static struct file_operations miscbeep_fops = {
 .owner = THIS_MODULE,
 .open = miscbeep_open,
 .write = miscbeep_write,
};
/* MISC设备结构体 */
static struct miscdevice beep_miscdev = {
 .minor = MISCBEEP_MINOR,
 .name = MISCBEEP_NAME,
 .fops = &miscbeep_fops,
};
/* flatform驱动的probe函数 */
static int miscbeep_probe(struct platform_device *dev){
```

```
 int ret = 0;
 printk("beep driver and device was matched!\r\n");
 /* 设置BEEP所使用的GPIO */
 /* 1、获取设备节点：beep */
 miscbeep.nd = of_find_node_by_path("/beep");
 if(miscbeep.nd == NULL) {
  printk("beep node not find!\r\n");
  return -EINVAL;
 }
 /* 2、 获取设备树中的gpio属性，得到BEEP所使用的BEEP编号 */
 miscbeep.beep_gpio = of_get_named_gpio(miscbeep.nd, "beep-gpio", 0);
 if(miscbeep.beep_gpio < 0) {
  printk("can't get beep-gpio");
  return -EINVAL;
 }
 /* 3、设置GPIO5_IO01为输出，并且输出高电平，默认关闭BEEP */
 ret = gpio_direction_output(miscbeep.beep_gpio, 1);
 if(ret < 0) {
  printk("can't set gpio!\r\n");
 }
 /* 不需要自己注册字符设备驱动，只需要注册misc设备驱动即可 */
 ret = misc_register(&beep_miscdev);
 if(ret < 0){
  printk("misc device register failed!\r\n");
  return -EFAULT;
 }

 return 0;
}
/* platform驱动的remove函数 */
static int miscbeep_remove(struct platform_device *dev){
 /* 注销设备的时候关闭LED灯 */
 gpio_set_value(miscbeep.beep_gpio, 1);
 /* 注销misc设备 */
 misc_deregister(&beep_miscdev);
 return 0;
}
/* 匹配列表 */
static const struct of_device_id beep_of_match[] = {
    { .compatible = "atkalpha-beep" },
    { /* Sentinel */ }
};
/* platform驱动结构体 */
static struct platform_driver beep_driver = {
     .driver     = {
         .name   = "imx6ul-beep",         /* 驱动名字，用于和设备匹配 */
         .of_match_table = beep_of_match, /* 设备树匹配表           */
     },
     .probe      = miscbeep_probe,
     .remove     = miscbeep_remove,
};
/* 驱动出口函数 */
static int __init miscbeep_init(void){
 return platform_driver_register(&beep_driver);
}
/* 驱动出口函数 */
static void __exit miscbeep_exit(void){
 platform_driver_unregister(&beep_driver);
}
```

```
module_init(miscbeep_init);
module_exit(miscbeep_exit);
MODULE_LICENSE("GPL");
```

2.3 测试程序编写

新建测试文件 miscbeepApp.c，并编写程序

```
#define BEEPOFF 0
#define BEEPON  1

int main(int argc, char *argv[]){
 int fd, retvalue;
 char *filename;
 unsigned char databuf[1];

 if(argc != 3){
  printf("Error Usage!\r\n");
  return -1;
 }

 filename = argv[1];
 fd = open(filename, O_RDWR); /* 打开beep驱动 */
 if(fd < 0){
  printf("file %s open failed!\r\n", argv[1]);
  return -1;
 }

 databuf[0] = atoi(argv[2]); /* 要执行的操作：打开或关闭 */
 retvalue = write(fd, databuf, sizeof(databuf));
 if(retvalue < 0){
  printf("BEEP Control Failed!\r\n");
  close(fd);
  return -1;
 }

 retvalue = close(fd); /* 关闭文件 */
 if(retvalue < 0){
  printf("file %s close failed!\r\n", argv[1]);
  return -1;
 }
 return 0;
}
```

2.4 运行测试

- 修改Makefile编译目标变量

```
obj-m := miscbeep.o
```

- 使用“make -j32”编译出驱动模块文件

```
make -j32
```

- 使用“arm-linux-gnueabihf-gcc”命令编译测试APP

```
arm-linux-gnueabihf-gcc miscbeepApp.c -o miscbeepApp
```

将驱动文件和APP可执行文件拷贝至“rootfs/lib/modules/4.1.15”中

使用“modprobe”命令加载驱动，加载成功后总线就会进行匹配

```
depmod  #第一次加载驱动时，需使用“depmod”命令
modprobe miscbeep.ko
```

- 加载成功后可以在/sys/class/misc 目录下看到一个名为“miscbeep”的子目录

```
/lib/modules/4.1.15 # ls /sys/class/misc/
autofs                  memory_bandwidth    rfkill
cpu_dma_latency         miscbeep            ubi_ctrl
fuse                    mxc_asrc            watchdog
hw_random               network_latency
loop-control            network_throughput
/lib/modules/4.1.15 #
```

> 所有的 misc 设备都属于同一个类， /sys/class/misc 目录下就是 misc 这个类的所有设备，每个设备对应一个子目录

- 驱动与设备匹配成功以后就会生成/dev/miscbeep 这个设备驱动文件

```
ls /dev/miscbeep -l  #查看设备的主次设备号
```

```
/lib/modules/4.1.15 # ls /dev/miscbeep -l
crw-rw----    1 0        0          10, 144 Jan  1 05:07 /dev/miscbeep
/lib/modules/4.1.15 #
```

- 使用以下命令打开或者关闭Beep

```
./miscbeepApp /dev/miscbeep 1
./miscbeepApp /dev/miscbeep 0
```