高级驱动学习方法

主讲老师： 计好奇

---

linux 字符设备驱动编程

1，实现入口函数 xxx_init()和卸载函数 xxx_exit()
2，申请设备号 register_chrdev_region (与内核相关)
3，注册字符设备驱动 cdev_alloc cdev_init cdev_add (与内核相关)
4，利用 udev/mdev 机制创建设备文件(节点) class_create, device_create (与内核相关)
5，硬件部分初始化
io 资源映射 ioremap,内核提供 gpio 库函数 (与硬件相关)
注册中断(与硬件相关)
初始化等待队列 (与内核相关)
初始化定时器 (与内核相关)
6，构建 file_operation 结构 (与内核相关)
7，实现操作硬件方法 xxx_open,xxx_read,xxxx_write...(与硬件相关)

想法： 有些通用代码不需要去写，内核帮我去写
只需要编写少部分代码（差异化代码）
目的： 修改/编写更少代码，去兼容更多是不同设备
代码重用，兼容强，可移植性

linux 程序框架的概念

1，内核引入程序框架思想:代码可重用,可维护,可伸缩
2，通用功能 写一次 可重用性好
3，差异功能 平台不同 可移植性好
4，内核框架采用分层
5，建立设备模型,它外在表现 平台设备驱动(总线,设备,驱动来实现的)
6，面向对象编程方式

面向对象代码实现

```
struct A{....}
struct B{
      struct A obj;//结构体,对象
}
```

B 是 A 的子类

分层代码的实现

1 定义结构体 抽象类

```
struct mydriver{
      char *name;
      int irq;
      int addr;
      void (*func)();
      struct mydriver *next;
```

```
}
```

2 初始化(底层硬件初始化)

```
构建mydriver对象
struct mydriver *drv=alloc(...);
设置mydriver对象
drv->irq=...
drv->addr=...
drv->func=key_func
```

注册

```
register(mydriver对象)
xxx_add(mydriver对象)
```

定义

```
void key_func(){...}
```

分层:

```
上层
struct mydriver *temp
tmp->name
tmp->func()
==================调用内核核心层接口
核心层: 全局变量,链表
struct mydriver *head
head->a->b;

================== 注册到内核核心层
底层
struct mydriver a;
a.name="key";
a.irq=EINTX
a.func=...
定义
void key_func(){...}
```

字符设备高级驱动课程

1,研究各种子系统(input,i2c,触摸屏,lcd)

2,不再是注册字符设备(通用层已经做好)

3,看手册操作寄存器所占比例少了.

4,大部分的搭框架,研究子系统找到最底层用平台设备驱动来实现

初级驱动和高级驱动的不同特点

1.控制器与外设关系复杂

2.代码量大,复杂 不从零开始写代码(参考内核提供代码,厂商提供代码),分析代码,看代码实现原理,移植代码

3.从宏观角度把握框架原理 如何实现,分层,要做什么?

4.软硬结合紧密,采用现成模板来实现

学习目标

1，熟悉各种设备驱动子系统特点(软件,硬件)

2，熟悉各种设备驱动的工作流程(框架实现原理)
3，采用模板学习如何移植各种设备驱动

学习方法：源码分析+接口驱动代码编程
1，控制器工作原理
2，分析驱动框架
3，研究 samsung 的驱动,看别人代码(提高代码阅读能力,分析能力)
4，移植驱动/写驱动