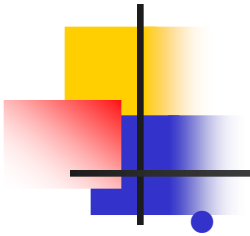
Ch4 内核与驱动简介

Enyi Tang

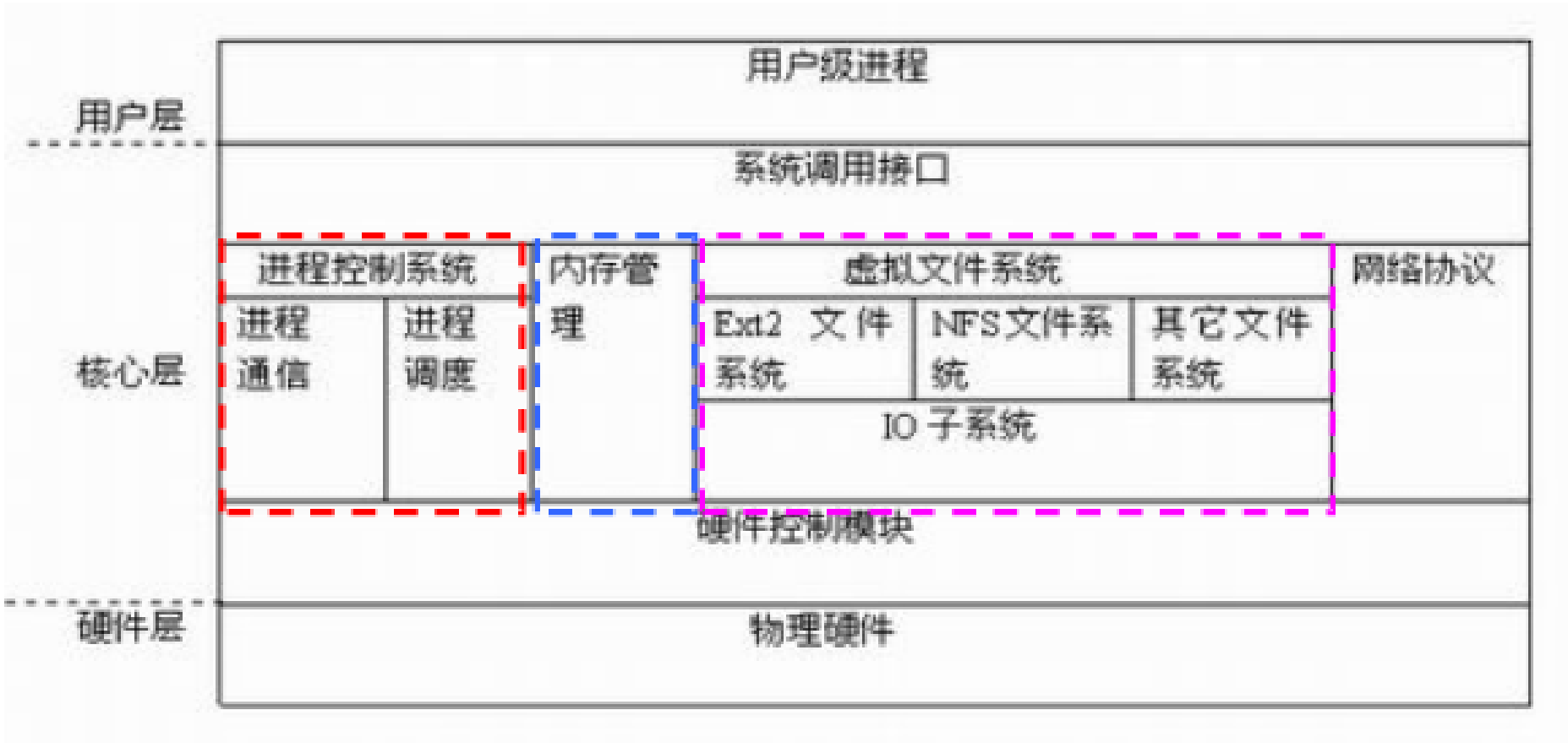
Software Institute of
Nanjing University



Linux内核简介

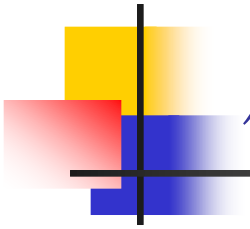
- 什么是内核
  - 操作系统是一系列程序的集合，其中最重要的部分构成了内核
  - 单内核/微内核
    - 单内核是一个很大的进程，内部可以分为若干模块，运行时是一个独立的二进制文件，模块间通讯通过直接调用函数实现
    - 微内核中大部分内核作为独立的进程在特权下运行，通过消息传递进行通讯
  - Linux内核的能力
    - 内存管理，文件系统，进程管理，多线程支持，抢占式，多处理支持
  - Linux内核区别于其他UNIX商业内核的优点
    - 单内核，模块支持
    - 免费/开源
    - 支持多种CPU，硬件支持能力非常强大
    - Linux开发者都是非常出色的程序员
    - 通过学习Linux内核的源码可以了解现代操作系统的实现原理

层次结构


用户级进程
用户层
系统调用接口
进程控制系统
进程通信
进程调度
内存管理
虚拟文件系统
Ext2 文件系统
NFS文件系统
其它文件系统
IO 子系统
网络协议
核心层
硬件控制模块
硬件层
物理硬件

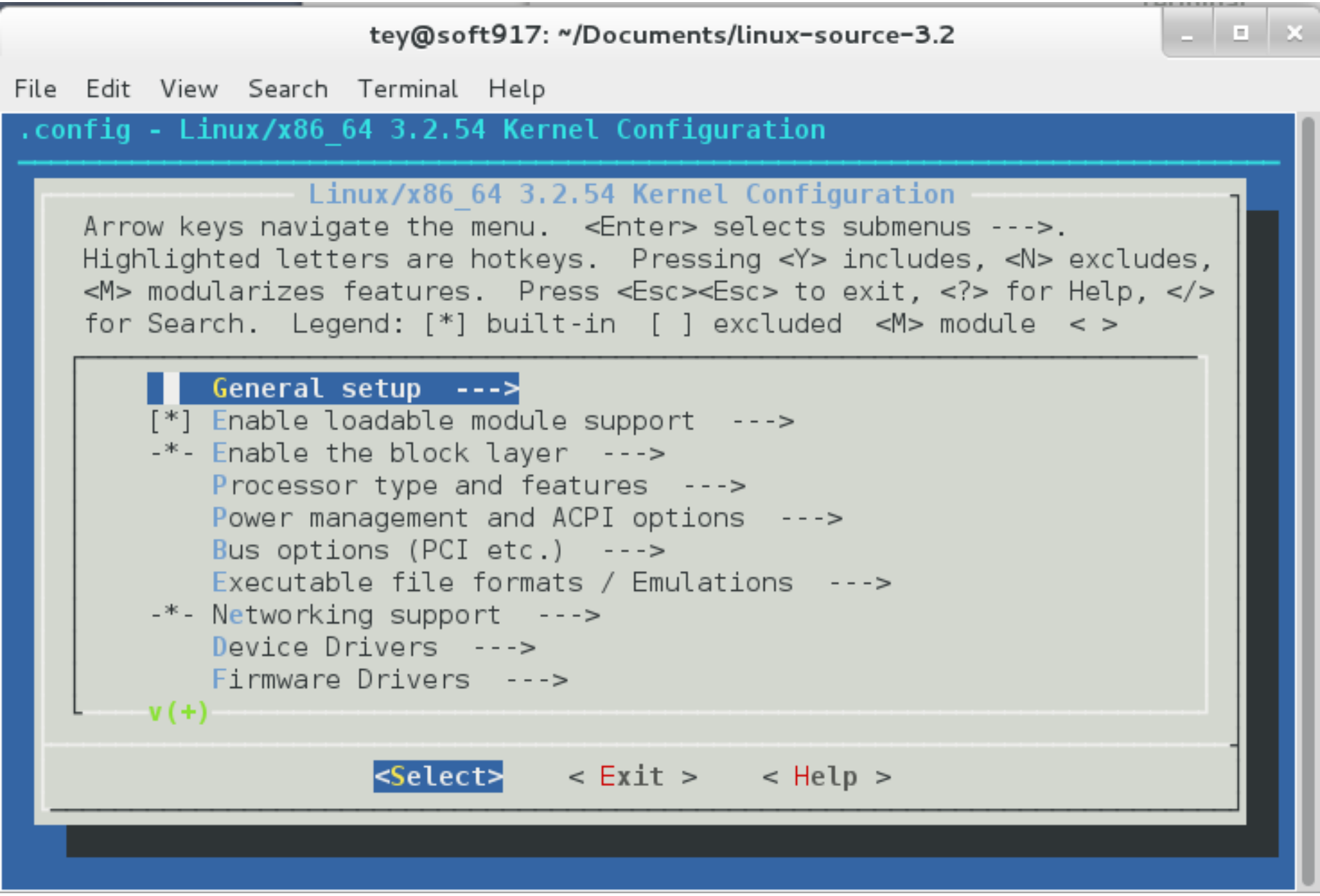
内核源代码获取

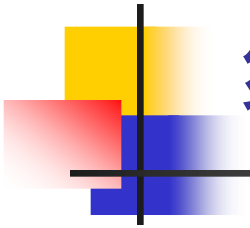
- https://www.kernel.org/
- apt-get方式
  - apt-cache search linux-source //查看内核版本
  - apt-get install linux-source-3.2
  - 下载下来的位置一般在/usr/src
- 从Ubuntu的源码库中获得内核源码
  - git clone git://kernel.ubuntu.com/ubuntu/ubuntu-hardy.git



后续操作

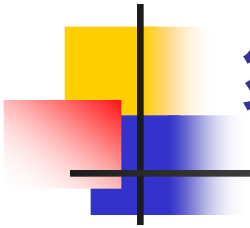
- 解压
  - tar jxvf /home/ldd/linux-3.2.tar.bz2
- 清除先前编译产生的目标文件
  - make clean
- 配置内核
  - make menuconfig


tey@soft917: ~/Documents/linux-source-3.2
File Edit View Search Terminal Help
.config - Linux/x86_64 3.2.54 Kernel Configuration
Linux/x86_64 3.2.54 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >
General setup --->
[*] Enable loadable module support --->
-*- Enable the block layer --->
Processor type and features --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Executable file formats / Emulations --->
-*- Networking support --->
Device Drivers --->
Firmware Drivers --->
v(+)
<Select> < Exit > < Help >



编译选项

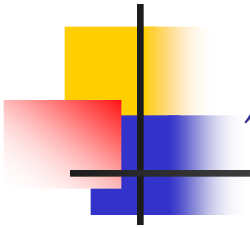
- 内核组件
  - Y(*) 要集成该组件
  - N( ) 不需要该组件，以后会没有这项功能
  - M 以后再加该组件为一个外部模块



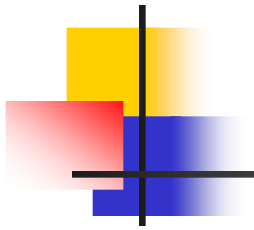
编译内核

- make
- make zImage
- make bzImage
- make modules

启用新内核



- make install (慎用)
  - 将编译好的内核copy到/boot
- 配置引导菜单



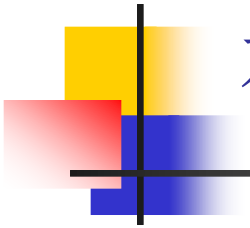
```
## ## End Default Options ##

title          Ubuntu 8.04.2, kernel 2.6.24-23-generic
root           (hd0,0)
kernel              /boot/vmlinuz-2.6.24-23-generic root=U
initrd              /boot/initrd.img-2.6.24-23-generic
quiet
```

```
title Win7
chainloader (hd0,0)/bootmgr
```

```
title win7
find --set-root  /bootmgr
chainloader  /bootmgr

title XP
find --set-root   /ntldr
chainloader  /ntldr
```

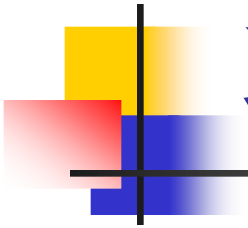


初始化程序的建立

- initrd
  - mkinitrd /boot/initrd.img $(uname -r)
- initramfs
  - mkinitramfs -o /boot/initrd.img 2.6.24-16
  - update-initramfs -u

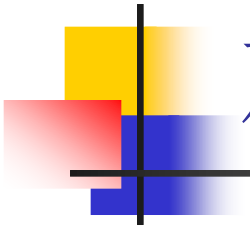
Debian和Ubuntu的简便办法

- make-kpkg
  - 用于make menuconfig之后
- 好处
  - 后面所有的部分自动做完
  - 会把编译好的内核打成deb安装包
  - 可以拷到其它机器安装



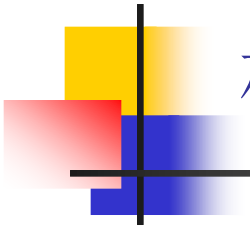
驱动

- 许多常见驱动的源代码集成在内核源码里
- 也有第三方开发的驱动，可以单独编译成模块.ko
- 编译需要内核头文件的支持



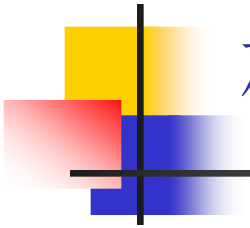
加载模块

- 底层命令
  - insmod
  - rmmod
- 高层命令
  - modprobe
  - modprobe -r



模块依赖

- 一个模块A引用另一个模块B所导出的符号，我们就说模块B被模块A引用。
- 如果要装载模块A，必须先要装载模块B。否则，模块B所导出的那些符号的引用就不可能被链接到模块A中。这种模块间的相互关系就叫做模块依赖。



模块的依赖

- 自动按需加载
- 自动按需卸载

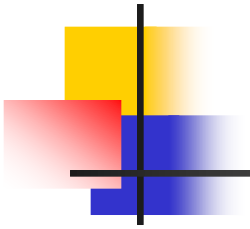
- moddep
- lsmod
- modinfo

模块之间的通讯

- 模块是为了完成某种特定任务而设计的。其功能比较的单一，为了丰富系统的功能，所以模块之间常常进行通信。其之间可以共享变量，数据结构，也可以调用对方提供的功能函数。

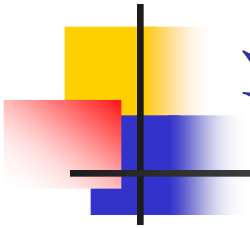
模块相关命令

- insmod <module.ko> [module parameters]
  - Load the module
  - 注意，只有超级用户才能使用这个命令
- rmmod
  - Unload the module
- lsmod
  - List all modules loaded into the kernel
  - 这个命令和cat /proc/modules等价
- modprobe [-r] <module name>
  - – Load the module specified and modules it depends



Linux内核模块与应用程序的区别

	C语言程序	Linux内核模块
运行	用户空间	内核空间
入口	main()	module_init()指定;
出口	无	module_exit()指定;
运行	直接运行	insmod
调试	gdb	kdbug, kdb, kgdb等



注意点

- 不能使用C库来开发驱动程序
- 没有内存保护机制
- 小内核栈
- 并发上的考虑

最简单的内核模块例子

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
static int __init hello_init(void)
{
        printk(KERN_INFO "Hello world\n");
        return 0;
}
static void __exit hello_exit(void)
{
        printk(KERN_INFO "Goodbye world\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

- static int __init hello_init(void)
- static void __exit hello_exit(void)
  - Static声明，因为这种函数在特定文件之外没有其它意义
  - __init标记，该函数只在初始化期间使用。模块装载后，将该函数占用的内存空间释放
  - __exit标记 该代码仅用于模块卸载。
- Init/exit
  - 宏：module_init/module_exit
  - 声明模块初始化及清除函数所在的位置
  - 装载和卸载模块时，内核可以自动找到相应的函数

    module_init(hello_init);

    module_exit(hello_exit);

编译内核模块

- Makefile文件

```
obj-m := hello.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
```

- Module includes more files

```
obj-m:=hello.o
hello-objs := a.o b.o
```

和硬件打交道

```
//file name: ioremap_driver.c
#include<linux/module.h>
#include<linux/init.h>
#include<asm/io.h>
//用于存放虚拟地址和物理地址
volatile unsigned long virt,phys;
//用与存放三个寄存器的地址
volatile unsigned long*GPBCON,*GPBDAT,*GPBUP;
void led_device_init(void){
    //0x56000010+0x10包揽全所有的IO引脚寄存器地址
    phys=0x56000010;
    // 0x56000010=GPBCON
    //在虚拟地址空间中申请一块长度为0x10的连续空间
    //这样，物理地址phys到phys+0x10对应虚拟地址
virt到virt+0x10
    virt=(unsigned long)ioremap(phys,0x10);
```

```
void led_device_init(void)
{
    // 0x56000010 + 0x10 包揽全所有的IO引脚寄存器地址
    phys=0x56000010 ;
    // 0x56000010=GPBCON
    //在虚拟地址空间中申请一块长度为0x10的连续空间
    //这样，物理地址phys到phys+0x10对应虚拟地址virt到virt+0x10
    virt=(unsigned long)ioremap(phys,0x10);
    GPBCON=(unsigned long*)(virt+0x00);
    //指定需要操作的三个寄存器的地址
    GPBDAT=(unsigned long*)(virt+0x04);
    GPBUP=(unsigned long*)(virt+0x08);
}
//led配置函数,配置开发板的GPIO的寄存器
void led_configure(void)
{
```

```
}
//led配置函数,配置开发板的GPIO的寄存器
void led_configure(void)
{

*GPBCON&=~(3<<10)&~(3<<12)&~(3<<16)&~(3<<20);
    //GPB12 defaule 清零
    *GPBCON|=(1<<10)|(1<<12)|(1<<16)|(1<<20);
    //output  输出模式
    *GPBUP|=(1<<5)|(1<<6)|(1<<8)|(1<<10);
    //禁止上拉电阻
}
//点亮led
void led_on(void)
{

*GPBDAT&=~(1<<5)&~(1<<6)&~(1<<8)&~(1<<10);
}
//灭掉led
void led_off(void)
```

```
*GPBDAT&=~(1<<5)&~(1<<6)&~(1<<8)&~(1<<10);
}
//灭掉led
void led_off(void)
{
    *GPBDAT|=(1<<5)|(1<<6)|(1<<8)|(1<<10);
}
//模块初始化函数
static int __init led_init(void)
{
    led_device_init();
    //实现IO内存的映射
    led_configure();
    //配置GPB5 6 8 10为输出
    led_on();
    printk("hello ON!\n");
    return 0 ;
}
//模块卸载函数
static void __exit led_exit(void)
```

```
    led_on();
    printk("hello ON!\n");
    return 0 ;
}
//模块卸载函数
static void __exit led_exit(void)
{
    led_off();
    iounmap((void*)virt);
    //撤销映射关系
    printk("led OFF!\n");
}
module_init(led_init);
module_exit(led_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("hurryliu<>");
MODULE_VERSION("2012-8-5.1.0");
```

End