

作者：道哥，10+年嵌入式开发老兵，专注于：C/C++、嵌入式、Linux。

关注下方公众号，回复【书籍】，获取 Linux、嵌入式领域经典书籍；回复【PDF】，获取所有原创文章(PDF 格式)。

目录

学习的困惑

实践环境

编译进内核

创建驱动程序目录

创建源文件

创建 Kconfig 文件

创建 Makefile 文件

编译

编译为驱动模块

编译所有的驱动模块

只编译 hello 这一个驱动模块

验证一下

资料下载

别人的经验，我们的阶梯！

大家好，我是道哥。今天给大家分享一些笔记本里的一些存货：Linux 系统中的驱动和中断相关。

大概会用 6~7 篇的文章，由浅入深的为大家介绍Linux 中驱动程序的编写方法。

文章的顺序，也是我之前自己学习时的顺序。

以前的学习记录比较零散，现在只是把它们按照一定的顺序重新梳理一下。

这几篇文章，理论知识会少一些，更注重实际的操作。

我会把操作引导的代码，全部上传到网盘上，在文末有下载说明。

只要根据文中介绍的步骤进行操作，就一定可以操作成功。

学习的困惑

记得以前我在开始学习驱动开发的时候，找来很多文章、资料来学习，但是总是觉得缺少了点全局视角。

就好像：我想看清一座山的全貌，但总是被困在一个、又一个山谷中一样。

主要的困惑有 3 点：

1. 每一篇文章的介绍都是正确的，但是如果把很多文章放在一起看，就会发现怎么说的都不一样啊？
2. 有些文章注重函数的介绍，但是缺乏一个全局的视角，从整体上来观察驱动程序的结构；
3. 对于一个新手来说，能够边学习、边实践，这是最好的学习方式，但是很多文章不会注意这方面。虽然文章内容很漂亮，但是不知道怎么去实践、验证。

因此，这几篇文章我们就从最简单的[驱动模块](#)编译开始，然后介绍[字符设备](#)驱动程序。

在这部分，会以 [GPIO](#) 为例子，重点描述其中的关键节点。

最后再介绍在[中断](#)处理程序中，如何利用[信号量](#)、[小任务](#)、[工作队列](#)，把内核事件传递到应用层来处理。

作为第一个开篇文章，从最简单的内核编译开始。

实际操作一下：如何把一个最简单的驱动程序(hello)，按照 2 种方式进行编译：

1. 编译进内核；
2. 编译为一个独立的驱动模块；

实践环境

为了便于测试，以下操作都是在 Ubuntu16.04 操作系统里完成的。

编译Linux驱动程序，肯定需要内核源码，这里选择的是 [linux-4.15](#) 版本，可以在官网下载。

文末有下载方式。

下载之后，把linux-4.15.tar.gz解压到Ubuntu中任意目录即可，例如：解压到~/tmp/目录下：

```
$ tar -zxvf linux-4.15.tar.gz -C ~/tmp/
```

编译进内核

创建驱动程序目录

linux中的驱动，一般都放在 [linux-4.15/drivers/](#) 目录下，因此在这个目录中创建一个hello文件夹。

```
$ mkdir linux-4.15/drivers/hello
```

对于一个驱动来说，最重要的就是3个文件：

1. 源代码
2. Kconfig
3. Makefile

只要按照固定的格式来编写这3个文件，linux内核的编译脚本就可以确保把我们的驱动程序编译进去。

创建源文件

首先是源码，在hello文件夹中创建源文件hello.c：

```
$ cd linux-4.15/drivers/hello
$ touch hello.c
```

源文件hello.c的内容是：

```
#include <linux/module.h>
#include <linux/init.h>

// 当驱动被加载的时候，执行此函数
static int __init hello_init(void)
{
    printk(KERN_ALERT "welcome, hello"\n");
    return 0;
}

// 当驱动被卸载的时候，执行此函数
static void __exit hello_exit(void)
{
    printk(KERN_ALERT "bye, hello\n");
}

// 版权声明
MODULE_LICENSE("GPL");

// 以下两个函数属于 Linux 的驱动框架，只要把驱动两个函数地址注册进去即可。
module_init(hello_init);
module_exit(hello_exit);
```

有两个小地方注意一下：

1. 在内核中，打印函数是 `printk`，而不是 `printf`;
2. 打印信息的级别有好几个，从 `DEBUG` 到 `EMERG`，这里使用的是 `KERN_ALERT`，方便查看打印信息。

创建 Kconfig 文件

这个文件是用来对内核进行配置的，当执行 `make menuconfig` 指令的时候，这个文件就被解析。

先创建文件：

```
$ cd linux-4.15/drivers/hello
$ touch Kconfig
```

添加如下内容：

```
config HELLO
tristate "hello driver"
help
    just a simplest driver.
default y
```

第一行内容 `config HELLO`，在执行配置的时候，将会生成一个变量 `CONFIG_HELLO`，而这个变量，将会在编译的时候，被 `Makefile` 引用。

最后一行的 `default y`，就表示把 `CONFIG_HELLO` 的值设置成 `y`，从而让这个驱动被编译到内核中。

现在，hello驱动中的KConfig配置文件已经准备好了，但是还需要这个配置文件登记到Linux内核的整体配置文件中。

也就是把它登记在 `linux-4.15/drivers/Kconfig` 文件的末尾：

```
source "drivers/hello/Kconfig"

endmenu    // 加在这一句的上面
```

现在，可以来执行下面指令，看一下具体的配置界面：

```
$ cd linux-4.15/
$ make distclean
$ make ARCH=x86_64 defconfig
$ make ARCH=x86_64 menuconfig
```

第2条指令，是用来把默认的配置保存到当前目录下的 `.config` 配置文件，也就是把一个默认的配置复制过来，作为我们自己的配置文件。

以后再修改配置参数时，修改的内容就会存储在 `.config` 文件中，

第3条指令，是用来配置内核的，可以进入 Device Drivers 菜单，然后在最底层看到我们的 `hello driver` 被标记成星号，
这表示被编译进内核。

```
Linux/x86_64 4.15.0 Kernel Configuration

Arrow keys navigate the menu. <Enter> selects submenus ---- (or empty 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 < > module capable

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 ---->
File systems ---->
Kernel hacking ---->
Security options ---->
-- Cryptographic API ---->
[*] Virtualization ---->
Library routines ---->

<Select> < Exit > < Help > < Save > < Load >
```

按向下方向键，把高亮定位到 `Device Drivers ---->`，然后敲回车键，进入到 `Device Drivers` 的配置界面。

按向下方向键，一直到最后一个条目，就可以看到我们的 `hello` 驱动了，如下：

```
Device Drivers

Arrow keys navigate the menu. <Enter> selects submenus ---- (or empty 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 < > module capable

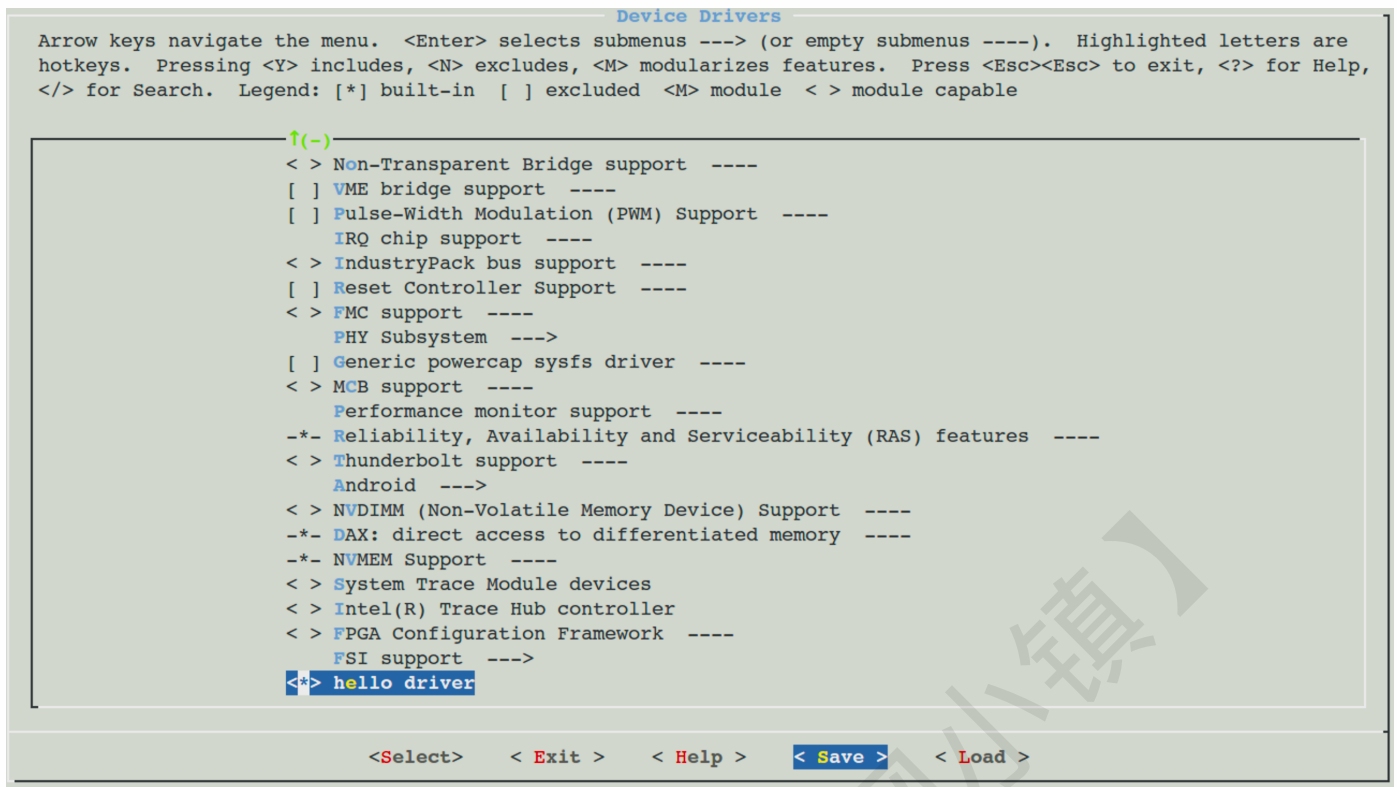
↑(-)
< > Non-Transparent Bridge support ----
[ ] VME bridge support ----
[ ] Pulse-Width Modulation (PWM) Support ----
IRQ chip support ----
< > IndustryPack bus support ----
[ ] Reset Controller Support ----
< > FMC support ----
PHY Subsystem ---->
[ ] Generic powercap sysfs driver ----
< > MCB support ----
Performance monitor support ----
-- Reliability, Availability and Serviceability (RAS) features ----
< > Thunderbolt support ----
Android ---->
< > NVDIMM (Non-Volatile Memory Device) Support ----
-- DAX: direct access to differentiated memory ----
-- NVMEM Support ----
< > System Trace Module devices
< > Intel(R) Trace Hub controller
< > FPGA Configuration Framework ----
FSI support ---->
<*> hello driver

<Select> < Exit > < Help > < Save > < Load >
```

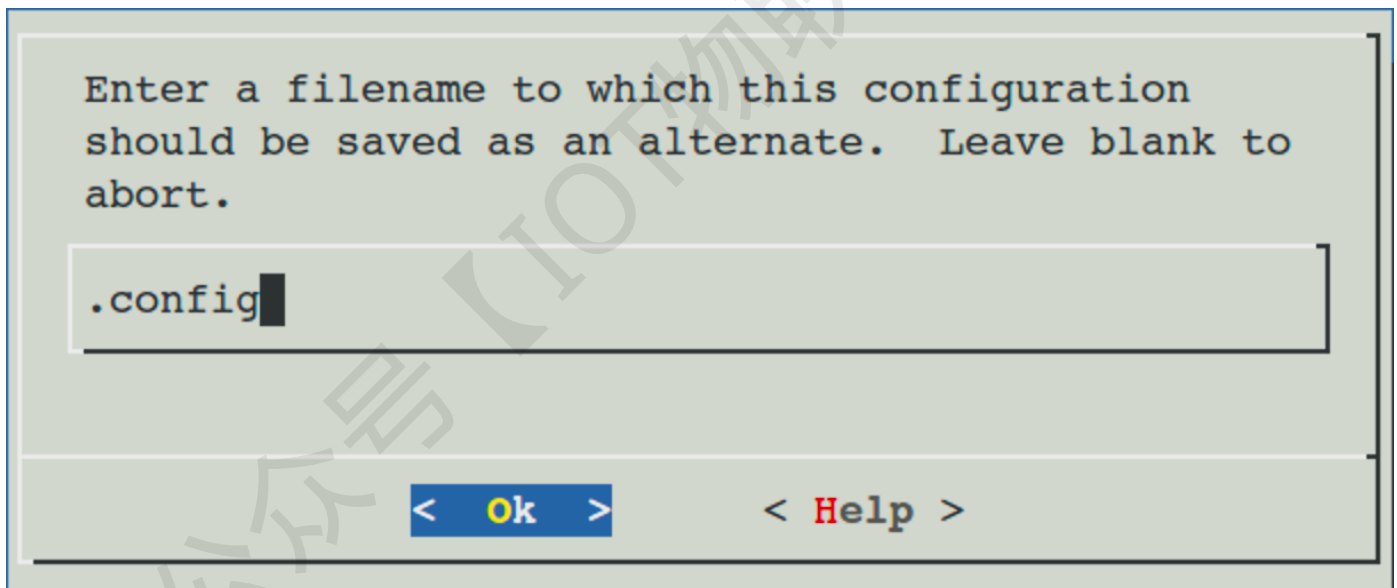
可以看到 `hello driver` 前面显示的是型号 `*`，这表示：该驱动将会编译进内核。

我们可以按下空格键试一下，会在三种标记中切换：型号，M，空值。M 标记意思是编译成驱动模块。

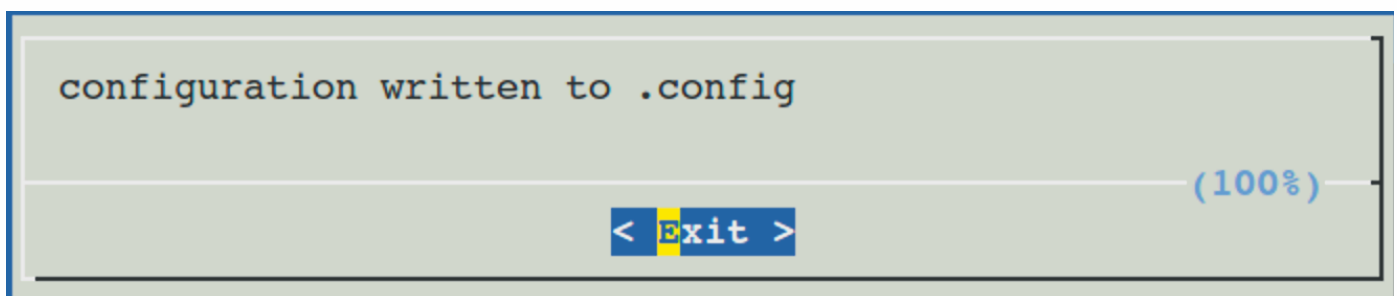
我们这里选择星号(编译进内核)，然后按下右方向键，最下方的几个按键的焦点移动到 按钮上：



按下回车键，就会弹出保存对话框，选择默认保存文件 .config 即可，然后在 按钮高亮的时候，按下回车键即可保存。



此时，在弹出的确认窗口中，选择<Exit>，按下回车键即可：



此时，返回到 Device Drivers 的配置界面，在最下面的按钮中，选择让<Exit>高亮，然后一路退出即可。

创建 Makefile 文件

Makefile文件是make工具的脚本，首先创建它：

```
$ cd linux-4.15/drivers/hello
$ touch Makefile
```

其中的内容只有一行：

```
obj-$(CONFIG_HELLO) += hello.o
```

1. CONFIG_HELLO 可以看做一个变量，在编译的时候，这个变量的值可能是：y, n 或者 m。
2. 在刚才的 Kconfig 参数配置中，CONFIG_HELLO 被设置为 y，于是这句话就被翻译成：obj-y += hello, 表示把 hello 驱动编译进内核。

现在，hello驱动程序的Makefile已经创建好了，我们还要让linux内核的编译框架知道这个文件才行。

在文件 [linux-4.15/drivers/Makefile](#) 中的末尾，添加如下内容：

```
obj-$(CONFIG_HELLO) += hello/
```

编译

万事俱备，只欠编译！依次执行如下指令：

```
$ cd linux-4.15/
$ make -j4
```

make指令执行结束之后，编译得到的内核中(vmlinux)就包含了我们的hello驱动。

编译为驱动模块

编译为驱动模块，也有两种操作方式：

编译所有的驱动模块

1. 在执行 make ARCH=x86_64 menuconfig 指令的时候，把 hello 配置成 M;
2. 然后在 linux-4.15 中执行编译模块指令：make -j4 modules。

编译成功之后，就可以得到文件：[linux-4.15/drivers/hello/hello.ko](#)。

这样的编译指令，是把所有的模块都编译了一次(在输出信息中，可以看到编译了很多模块)。

只编译 hello 这一个驱动模块

另外一种编译驱动模块的方式是：进入hello目录，只编译这一个驱动模块。

这种编译方法，就需要修改hello目录下的Makefile文件了，内容如下：

可以把 hello 目录下的所有文件删除，只保留源文件 hello.c，然后新建 Makefile 文件。

```
ifneq ($(KERNELRELEASE),)
    obj-m := hello.o
else
    KERNELDIR ?= /lib/modules/$(shell uname -r)/build
    PWD := $(shell pwd)
default:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
clean:
    $(MAKE) -C $(KERNEL_PATH) M=$(PWD) clean
endif
```

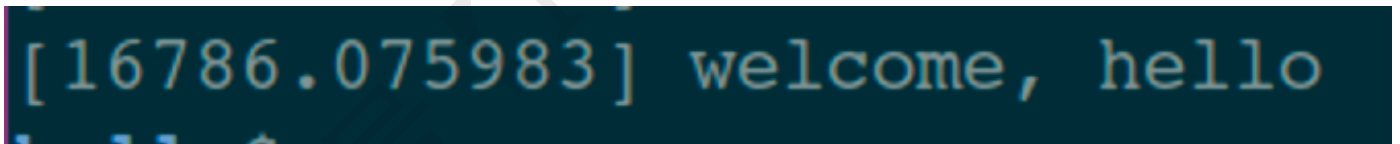
然后，在hello文件夹中执行make指令，即可得到驱动模块 [hello.ko](#)。

验证一下

加载驱动：

```
$ cd linux-4.15/drivers/hello
$ sudo insmod ./hello.ko
```

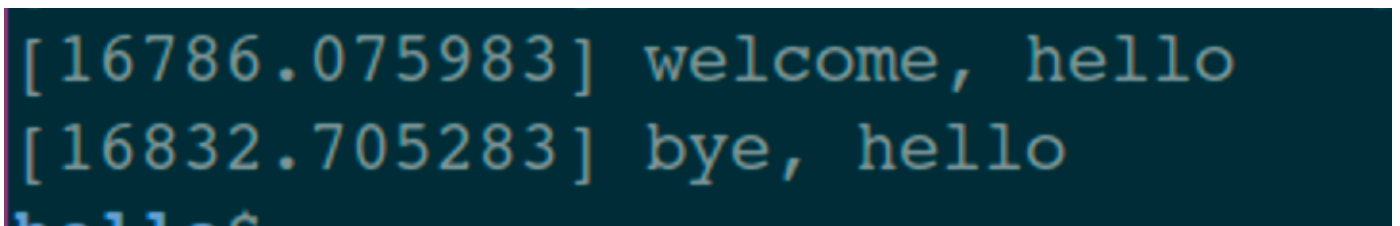
此时终端窗口是没有任何输出的，需要输入指令 [dmesg | tail](#)，可以看到 [hello_init](#) 函数的输出内容：



卸载驱动：

```
$ sudo rmmod hello
```

再次输入 [dmesg | tail](#)，可以看到 [hello_exit](#) 函数的输出内容：



资料下载

在公众号【IOT物联网小镇】的后台回复关键字：1112，获取下列文件的网盘地址：

linux-4.15.tar.gz

hello文件夹压缩包

----- End -----

推荐阅读

【1】《Linux 从头学》系列文章

【2】C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻

【3】原来gdb的底层调试原理这么简单

【4】内联汇编很可怕吗？看完这篇文章，终结它！

其他系列专辑：[精选文章](#)、[应用程序设计](#)、[物联网](#)、[C语言](#)。



微信搜一搜



IOT物联网小镇

星标公众号，第一时间看文章！

C/C++、物联网、嵌入式、Lua语言
Linux 操作系统、应用程序开发设计



扫码关注公众号



道哥 个人微信

喜欢请**分享**，满意点个**赞**，最后点**在看**。