目录

[一、 helloworld驱动实验 1](#_Toc163406200)

[1. 内核模块代码框架组成 1](#_Toc163406201)

[2. printk 函数 1](#_Toc163406202)

[3. 许可证 1](#_Toc163406203)

[4. 相关信息声明 2](#_Toc163406204)

[5. makefile 2](#_Toc163406205)

[6. helloworld.c 3](#_Toc163406206)

[7. 使用模块 3](#_Toc163406207)

# helloworld驱动实验

## 内核模块代码框架组成

* 模块加载函数 (必须)： 当通过 insmod 或 modprobe 命令加载内核模块时，模块的加载函数就会自动被内核执行，完成本模块相关的初始化工作。
* 模块卸载函数 (必须)： 当执行 rmmod 命令卸载模块时，模块卸载函数就会自动被内核自动执行，完成相关清理工作。
* 模块许可证声明 (必须)： 许可证声明描述内核模块的许可权限，如果模块不声明，模块被加载时，将会有内核被污染的警告。
* 模块参数： 模块参数是模块被加载时，可以传值给模块中的参数。
* 模块导出符号： 模块可以导出准备好的变量或函数作为符号，以便其他内核模块调用。
* 模块的其他相关信息： 可以声明模块作者等信息。

## printk 函数

* printk：内核模块无法使用 glibc 库函数中的printf()函数，内核自身实现的一个类 printf 函数，但是需要指定打印等级。

– #define KERN\_EMERG “<0>”通常是系统崩溃前的信息

– #define KERN\_ALERT “<1>”需要立即处理的消息

– #define KERN\_CRIT “<2>”严重情况

– #define KERN\_ERR “<3>”错误情况

– #define KERN\_WARNING “<4>”有问题的情况

– #define KERN\_NOTICE “<5>”注意信息

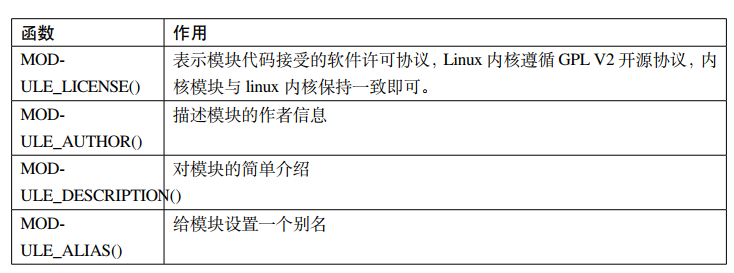
– #define KERN\_INFO “<6>”普通消息

– #define KERN\_DEBUG “<7>”调试信息

## 许可证

Linux 是一款免费的操作系统，采用了 GPL 协议，允许用户可以任意修改其源代码。内核模块许可证有“GPL”，“GPL v2”，“GPL and additional rights”，“Dual SD/GPL”，“DualMPL/GPL”，“Proprietary”。

## 相关信息声明



## makefile

# 内核编译输出的目录

KDIR:=/home/tiam/linuxLearn/driver/kernel/rk3568Kernel/kernel

# arm64位环境

ARCH=arm64

# 编译工具链

CROSS\_COMPILE=aarch64-linux-gnu-

# 导出环境变量

**export** ARCH CROSS\_COMPILE

# helloworld.c对应的.o文件

obj-m += helloworld.o

# 获取当前目录的环境变量

PWD?=$(shell pwd)

# make

all:

$(MAKE) -C $(KDIR) M=$(CURDIR) modules

# clean

.PHONE:clean copy

clean:

$(MAKE) -C $(KDIR) M=$(CURDIR) clean

copy:

sudo cp \*.ko /home/tiam/lubanCatDir

‘$(MAKE)modules’实际上是执行 Linux 顶层 Makefile 的伪目标 modules。通过选项‘-C’，可以让 make 工具跳转到源码目录下读取顶层 Makefile。‘M=$(CURDIR)’表明返回到当前目录，读取并执行当前目录的 Makefile，开始编译内核模块。 CURDIR 是 make 的内嵌变量，自动设置为当前目录。

## helloworld.c

#include <linux/module.h>

#include <linux/init.h>

#include <linux/kernel.h>

//驱动入口函数

**static** **int** \_\_init hello\_init(**void**){

printk(KERN\_EMERG "[KERN\_EMERG] Hello World Module Init\n");

printk( "[default] Hello World Module Init\n");

**return** 0;

}

//驱动出口函数

**static** **void** \_\_exit hello\_exit(**void**){

printk("[default] goodbye\n");

}

//注册入口和出口函数

module\_init(hello\_init);

module\_exit(hello\_exit);

//协议注册、作者声明、模块描述、模块别名

MODULE\_LICENSE("GPL2");

MODULE\_AUTHOR("tiam");

MODULE\_DESCRIPTION("hello world module");

MODULE\_ALIAS("test module");

## 使用模块

* lsmod

lsmod 列出当前内核中加载的模块，格式化显示在终端，其原理就是将/proc/modules 中的信息调整一下格式输出。 lsmod 输出列表有一列 Used by，它表明此模块正在被其他模块使用，显示了模块之间的依赖关系。

* insmod

如果要将一个模块加载到内核中， insmod 是最简单的办法， insmod+ 模块完整路径就能达到目的，前提是你的模块不依赖其他模块，还要注意需要 sudo 权限。如果你不确定是否使用到其他模块的符号，你也可以尝试 modprobe，后面会有它的详细用法。通过 insmod 命令加载 hellomodule.ko 内存模块加载该内存模块的时候，该内存模块会自动执行module\_init() 函数，进行初始化操作，该函数打印了‘hello module init’。

* modprobe

modprobe 和 insmod 具备同样的功能，同样可以将模块加载到内核中，除此以外 modprobe 还能检查模块之间的依赖关系，并且按照顺序加载这些依赖，可以理解为按照顺序多次执行 insmod。

* depmod

modprobe 是怎么知道一个给定模块所依赖的其他的模块呢？在这个过程中， depend 起到了决定性作用，当执行 modprobe 时，它会在模块的安装目录下搜索 module.dep 文件，这是 depmod 创建的模块依赖关系的文件。

* rmmod

rmod 工具仅仅是将内核中运行的模块删除，只需要传给它路径就能实现。rmmod 命令卸载某个内存模块时，内存模块会自动执行 \*\_exit() 函数，进行清理操作，我们的hellomodule 中的 \*\_exit() 函数打印了一行内容，但是控制台并没有显示，可以使用 dmesg 查看，之所以没有显示是与 printk 的打印等级有关。 rmmod 不会卸载一个模块所依赖的模块，需要依次卸载，当然使用modprobe -r 可以一键卸载。

* modinfo

modinfo 用来显示我们在内核模块中定义的几个宏。我们可以通过 modinfo 来查看 hellomodule，我们从打印的输出信息中，可以了解到，该模块遵循的是 GPL 协议，该模块的作者是 embedfire，该模块的 vermagic 等等。而这些信息在模块代码中由相关内核模块信息声明函数声明