

实验 02 HelloDriverModule

2.1 本章导读

本实验将带您走进 Linux 设备驱动的精彩世界。

Linux 设备驱动会以模块的形式出现，所以学会编写、编译、加载、卸载 Linux 内核模块是学习 Linux 驱动的先决条件。

本期实验 HelloDriverModule，以一个简单的 Linux 驱动为例，实现打印功能，让用户对 Linux 驱动模块有一个基本认识。

2.1.1 工具

2.1.1.1 硬件工具

- 1) iTOP4412 开发板
- 2) U 盘或者 TF 卡
- 3) PC 机
- 4) 串口

2.1.1.2 软件工具

- 1) 虚拟机 Vmware
- 2) Ubuntu12.04.2
- 3) 超级终端 (串口助手)

4) 源码文件夹 “HelloDriverModule”

2.1.2 预备课程

1) Linux 无界面最小系统

2) Linux 内核的编译

3) Linux 常用命令

2.1.3 视频资源

本节配套视频为 “视频 02_HelloDriverModule”

2.2 学习目标

本章需要学习以下内容：

Vim 编辑器显示中文字符

掌握 Linux 驱动的最简结构

学习以模块化形式编译驱动的方法

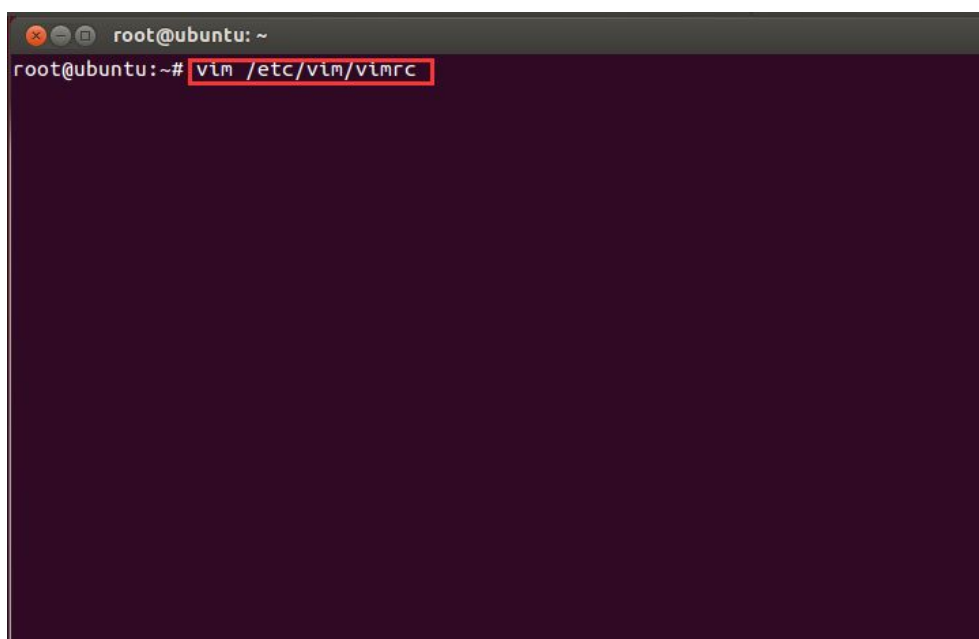
学习如何加载驱动、查看驱动、卸载驱动

2.3 Vim 显示中文字符

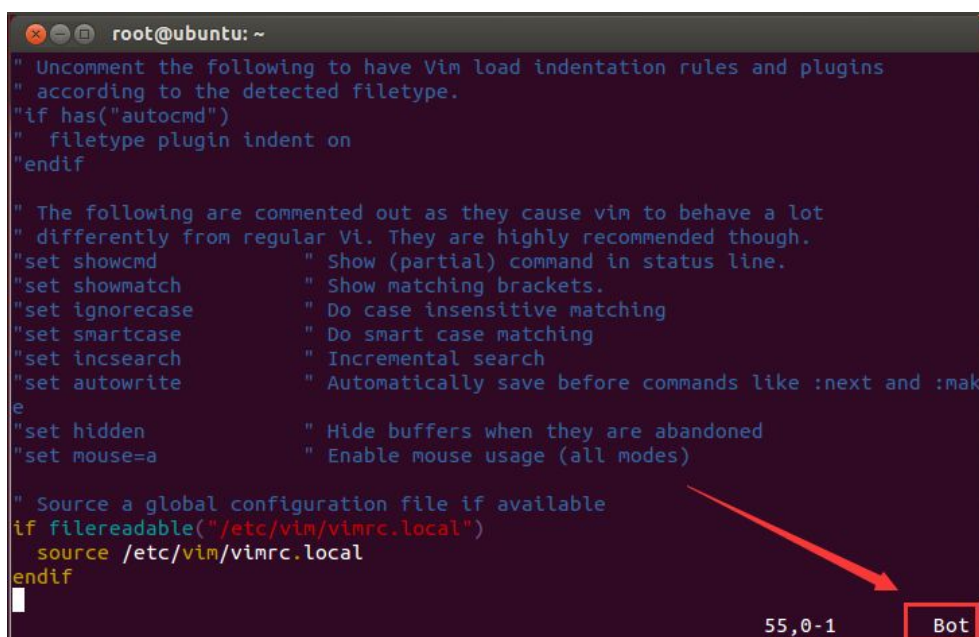
如果大家在默认的 Ubuntu 系统下使用 Vim 编辑器查看源码，中文字符会显示为乱码。

下面给大家简单介绍一下如何正常显示中文。

打开 vim 编辑器的配置文件 “/etc/vim/vimrc” ，使用命令 “vim /etc/vim/vimrc” ，如下图所示。



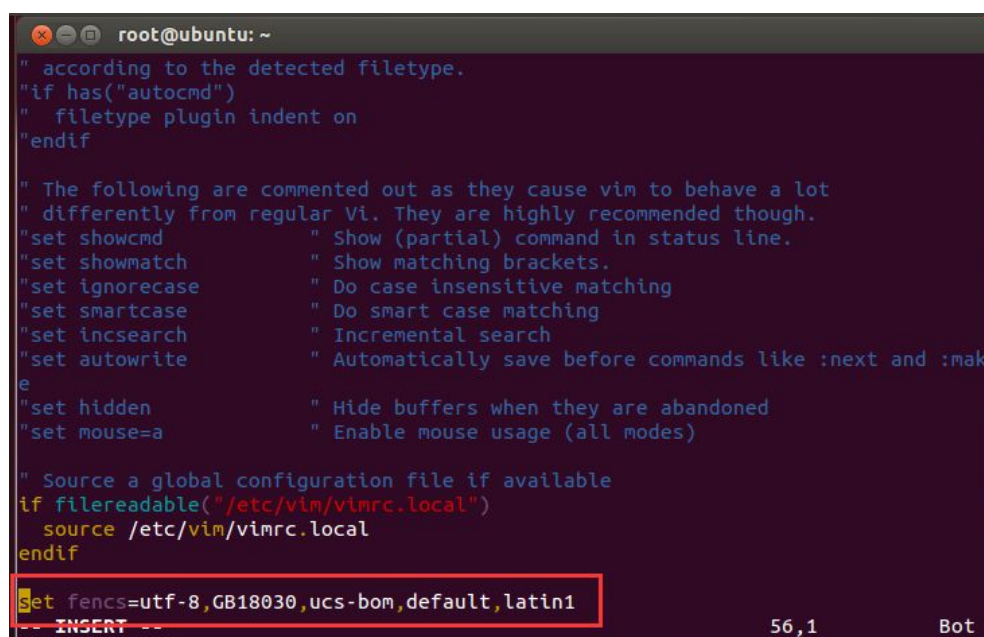
如下图所示，进入到最底行，右下角会显示 “Bot” 。



在文件 “/etc/vim/vimrc” 最底端添加以下代码。

```
set fencs=utf-8,GB18030,ucs-bom,default,latin1
```

添加完成之后，如下图所示。



```
root@ubuntu: ~  
" according to the detected filetype.  
" if has("autocmd")  
"   filetype plugin indent on  
"endif  
  
" The following are commented out as they cause vim to behave a lot  
" differently from regular Vi. They are highly recommended though.  
"set showcmd           " Show (partial) command in status line.  
"set showmatch         " Show matching brackets.  
"set ignorecase        " Do case insensitive matching  
"set smartcase         " Do smart case matching  
"set incsearch         " Incremental search  
"set autowrite         " Automatically save before commands like :next and :mak  
e  
"set hidden            " Hide buffers when they are abandoned  
"set mouse=a          " Enable mouse usage (all modes)  
  
" Source a global configuration file if available  
if filereadable("/etc/vim/vimrc.local")  
    source /etc/vim/vimrc.local  
endif  
  
set fencs=utf-8,GB18030,ucs-bom,default,latin1  
-- INSERT --
```

最后保存退出，Vim 编辑器就可以正常显示中文字符。

如下图所示，下面截图的为本节的驱动代码，大家可以看到中文字符可以正常显示。这一步不用测试，到了本章节后面的时候可以使用 Vim 编辑器测试一下。



```
#include <linux/init.h>  
/*包含初始化宏定义的头文件,代码中的module_init和module_exit在此文件中*/  
  
#include <linux/module.h>  
/*包含初始化加载模块的头文件,代码中的MODULE_LICENSE在此头文件中*/  
  
MODULE_LICENSE("Dual BSD/GPL");  
/*声明是开源的, 没有内核版本限制*/  
MODULE_AUTHOR("iTOPEET_dz");  
/*声明作者*/  
  
static int hello_init(void)  
{  
    printk(KERN_EMERG "Hello World enter!\n");  
    /*打印信息, KERN_EMERG表示紧急信息*/  
    return 0;  
}
```

2.4 Linux 内核最小模块代码分析

Linux 内核的整体框架非常大，提供给大家的内核源码，大家应该见识过了，包含了各种各样的组件，而且有 1 万多个文件，那么这些文件是怎么添加到内核中的呢？

Linux 内核针对驱动的处理有以下两种方式：

第一种方式：把所有需要的功能全部编译到内核中，这种方式会导致重新添加或者删除功能的时候，需要重新编译内核。

第二种方式：动态的添加模块，也就是这个实验要介绍的“模块的方式添加驱动”。

如下图所示，这是一个非常简单的驱动代码。源码文件“mini_linux_module.c”大家可以在视频目录“02_HelloDriverModule”中找到。

mini_linux_module.c 中代码如下。

```
#include <linux/init.h>
/*包含初始化宏定义的头文件,代码中的module_init和module_exit在此文件中*/
#include <linux/module.h>
/*包含初始化加载模块的头文件,代码中的MODULE_LICENSE在此头文件中*/
```

```
MODULE_LICENSE("Dual BSD/GPL");
/*声明是开源的,没有内核版本限制*/
MODULE_AUTHOR("iTOPEET_dz");
/*声明作者*/
```

声明模块信息

头文件

```
static int hello_init(void)
{
    printk(KERN_EMERG "Hello World enter!\n");
    /*打印信息, KERN_EMERG表示紧急信息*/
    return 0;
}

static void hello_exit(void)
{
    printk(KERN_EMERG "Hello world exit!\n");
}
```

功能区

```
module_init(hello_init);
/*初始化函数*/
module_exit(hello_exit);
/*卸载函数*/
```

模块的入口

模块的出口

这个内核模块只有简单的几个功能,加载模块、卸载模块、对 GPL 的声明、描述信息例如作者以及加载和卸载时候的打印函数。

可以看到这个文件中的代码被红色框图划分为 5 大块,分别是:

第一部分:必须包含的头文件 linux/init.h 和 linux/module.h,想要编译成模块就必须使用这个两个头文件。

第二部分:驱动申明区。

在所有的声明中下面这一句最重要。

```
MODULE_LICENSE("Dual BSD/GPL");
```

```
/*声明是开源的，没有内核版本限制*/
```

这一句代码声明你遵循 GPL 协议，否则模块在加载的时候，内核会提示被污染的“警报”。

```
MODULE_AUTHOR("iTOPEET_dz");
```

```
/*声明作者*/
```

作者的声明，声明这个驱动代码是谁写的。这个申明不是必须的。

第三部分：功能区代码。

在功能区里面，加载驱动和卸载驱动时候调用的函数，这两个函数都只是调用了 printk 函数。

第四部分：模块的入口。

加载模块。采用动态的方式添加驱动到内核中，添加驱动的入口就是这个函数。

加载的时候调用了功能区的函数 static int hello_init(void)

这里特别提醒只接触过单片机用户或者那些没有接触过操作系统的用户，你会发现这里并没有 main 函数。

Linux 操作系统相当于“一个球”，要做的事情就是在这个球上添加驱动来实现具体的功能，不用去管这个球是从哪里开始旋转，转到什么地方了。更简单的理解就是，Linux 只是一个工具，学会使用就可以了，就像学习汽车驾驶，没有教练会从发动机原理开始介绍，只会教“方向盘右转一圈”“方向盘左转一圈”“拉手刹”“换挡”之类的。

如果你非要找一个 main 函数才罢休，那么你可以把模块的初始化函数当做 main 函数，也是模块的入口。

第五部分：模块的出口。

卸载模块。采用动态添加驱动的方式来加载驱动，那么也可以动态的卸载这个驱动，执行的是和添加驱动相反的一个过程。

卸载的时候调用了功能区的函数 `static void hello_exit(void)`

编译模块的时候会生成一个 ko 文件，可以使用 `insmod` 和 `rmmod` 加载和卸载它。加载的时候会打印 “Hello World enter!” 卸载的时候会打印 “Hello world exit!”。

这里针对打印函数 `printk` 做一个简单说明，`printk` 函数是在内核中使用的，不是用户空间的 `printf` 函数，不过用法类似。

这个函数在写驱动的时候，几乎都会用到，这是一个必须掌握的调试方法，也是最基本的方法。在后面的例子中，会慢慢的介绍到其它的调试方法，到后期再给大家做个调试 Debug 方法的总结。因为最终的目标是用户能够自己写代码和调试驱动，那么这个 Debug 能力是一定要具备的。

打印函数 `printk` 向超级终端传递数据，`KERN_EMERG` 是紧急情况的标识，加载和卸载驱动模块的时候，分别打印输出 “Hello World enter!” “Hello world exit!”。

打印函数 `printk` 是分级的，它的 8 个级别如下：

```
#define KERN_EMERG 0
```

```
/*紧急事件消息，系统崩溃之前提示，表示系统不可用*/
```

```
#define KERN_ALERT 1
```



```
/*报告消息，表示必须立即采取措施*/

#define KERN_CRIT 2

/*临界条件，通常涉及严重的硬件或软件操作失败*/

#define KERN_ERR 3

/*错误条件，驱动程序常用 KERN_ERR 来报告硬件的错误*/

#define KERN_WARNING 4

/*警告条件，对可能出现问题的情况进行警告*/

#define KERN_NOTICE 5

/*正常但又重要的条件，用于提醒*/

#define KERN_INFO 6

/*提示信息，如驱动程序启动时，打印硬件信息*/

#define KERN_DEBUG 7

/*调试级别的消息*/
```

2.4 Linux 内核模块结构

一个 Linux 内核模块主要由以下几个部分组成。

1) 模块加载函数

当通过 insmod 命令加载内核模块的时候，模块的加载函数会自动被调用到内核运行，完成模块的初始化工作

2) 模块卸载函数

当通过 `rmmod` 命令卸载内核模块的时候，模块的卸载函数会自动被调用到内核运行，完成模块的卸载工作，完成与模块卸载函数相反的功能。

3) 模块的许可声明

LICENSE 声明描述内核的许可权限，如果不声明 LICENSE，模块被加载的时候，会受到内核被污染的警告。

Linux 在使用的过程中，需要接受 GPL 协议，体现在代码中就是添加类似

`MODULE_LICENSE("Dual BSD/GPL");`的语句

在 Linux 内核模块领域，可接受的 LICENSE 参数包括 "GPL"、"GPL v2"、"GPL and additional"、"Dual BSD/GPL"、"Dual MPL/GPL" 等，当参数是前面的几个时，那么就表明你遵循 GPL 协议。

还有一个参数 "Proprietary"，这个参数表明是私有的，除非你的模块显式地声明一个开源版本，否则内核也会默认你这是一个私有的模块 (Proprietary)。

4) 模块作者信息等

体现在代码中，就是类似 `MODULE_AUTHOR("iTOPEET_dz");`的语句。其中 `iTOPEET_dz` 是参数。

小贴士：GPL 协议简介

GPL，是 General Public License 的缩写，是一份 GNU 通用公共授权非正式的中文翻译。GPL 协议中一个很核心的内容是：如果你接受这个协议，那么你就可以免费使用 Linux 中的代码，当你免费使用 Linux 的代码开发出新的代码，那么你就应该以源码或者二进制文件的方式

免费发布；如果你不接受这个协议，那么你就无权使用 Linux 源码。更加详细的内容，读者可以上网查一下，关键词是“Linux GPL 协议”。

2.5 模块加载函数

模块的加载函数“`module_init(function)`”，返回整数型，如果执行成功，则返回 0。否则返回错误信息。

有时候芯片供应商并不提供芯片驱动的源码，只是提供驱动 module 的 ko 文件，这个时候就需要调用 `request_module(module_name)` 来加载驱动。

在 Linux 中，标示为 `_init` 的函数都是初始化函数，这些函数占用的内存空间，在 Linux 启动或者模块加载初始化之后，是会被释放掉的。除了函数，数据也是可以定义为“`_数据`”，这些数据在 linux 启动或者或者模块加载初始化之后，也是会被释放掉。这一部分的知识，在后面会逐渐使用到。

2.6 模块卸载函数

模块卸载就是模块加载的“逆向过程”，比较容易理解。

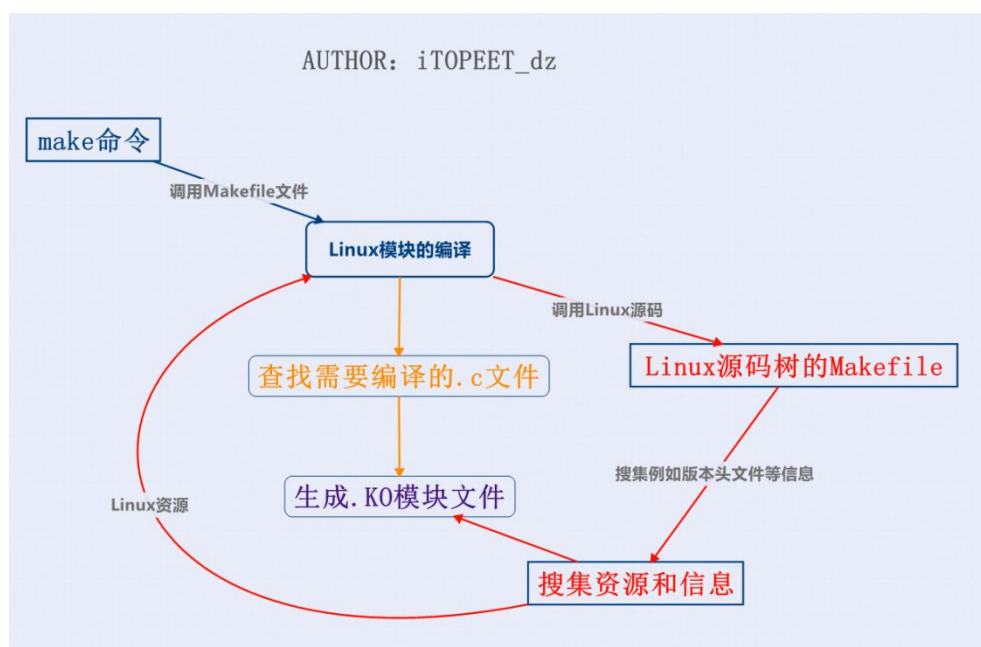
模块的卸载函数“`module_init(function)`”不返回任何值。

一般说来，在卸载函数中要完成和加载函数中相反的功能，例如你调用了系统或者硬件的资源，那么你在卸载函数中，就需要释放掉。

2.7 模块编译的流程

Linux 模块一般是使用脚本语言来编译，脚本语言的种类非常多，语法丰富，不过只需要学会使用即可，能够仿照着写就可以了，这并不影响开发。

如下图所示，模块的编译流程图。



通过上图可以看到编译中，当执行执行 make 命令之后，调用 Makefile 文件进行 Linux 模块的编译。

Linux 模块的编译分为两个条线。

第一条红色的线：进入 Linux 源码中，调用版本信息以及一些头文件等。

这一条线经过的是整个 Linux 的源码文件。

第二条橙色的线：搜集完 Linux 源码树的信息之后，Makefile 继续执行，调用编译.KO 文件的源码文件，这里是 mini_linux_module.c 这个 “iTop4412_Kernel_3.0” 整个源码。

这一条线走的是 mini_linux_module.c，虽然都是源码，但是此源码非彼源码。

Makefile 文件通过执行上面两条线，通过搜集到信息，最终编译生成.KO 模块

这里要学习和理解的重点就是，编译模块也必须用到内核的源码，因为这涉及到内核版本以及头文件。如果版本不对，那么模块有可能无法加载和运行；如果没有头文件，编译就无法通过。

2.8 脚本文件 Makefile

在单片机或者上位机编程的时候，都有集成开发工具。程序员按照开发工具的规则，将代码放入指定的位置，通常是一个 main.c 文件加上很多.c 文件，代码写好了，开发工具中某个按钮一点，就给你自动编译成了二进制文件。

在 Linux 中，并没有这样的集成开发工具，这里还需要自己写编译文件 Makefile。

编译文件一般是用脚本编写，脚本成千上万，脚本语言学也学不完，脚本的学习最好是用到哪里学到哪。

Makefile 编译文件如下，下面的这个文件可以在视频目录“02_HelloDriverModule”找到。

```
#!/bin/bash
#通知编译器我们要编译模块的哪些源码
#这里是编译itop4412_hello.c这个文件编译成中间文件mini_linux_module.o
obj-m += mini_linux_module.o

#源码目录变量，这里用户需要根据实际情况选择路径
#作者是将Linux的源码拷贝到目录/home/topeet/android4.0下并解压的
KDIR := /home/topeet/android4.0/iTop4412_Kernel_3.0

#当前目录变量
PWD ?= $(shell pwd)

#make命名默认寻找第一个目标
#make -C就是指调用执行的路径
#$(KDIR) Linux源码目录，作者这里指的是/home/topeet/android4.0/iTop4412_Kernel_3.0
#$(PWD) 当前目录变量
#modules要执行的操作
all:
    make -C $(KDIR) M=$(PWD) modules

#make clean执行的操作是删除后缀为o的文件
clean:
    rm -rf *.o
```

如上图所示，就是编译 mini_linux_module 的脚本文件。下面详细的介绍每一句的含义。

`#!/bin/bash`

通知编译器这个脚本使用的是那个脚本语言

`obj-m += mini_linux_module.o`

这是一个标准用法，表示要将 mini_linux_module.c 文件编译成 mini_linux_module.o 文件，如果还需要编译其它的文件，则在后面添加即可。

`KDIR := /home/topeet/android4.0/iTop4412_Kernel_3.0`

这一行代码表示内核代码的目录，如果没有内核源码，那么模块的编译无法进行，因为会缺乏版本支持和头文件。KDIR 是一个变量。

`PWD ?= $(shell pwd)`

这一句是提供一个变量，然后将当前目录的路径传给这个变量。pwd 是一个命令，表示当前目录，PWD 是一个变量。

all:

```
make -C $(KDIR) M=$(PWD) modules
```

在使用执行脚本编译命令“make”的时候，它会默认来寻找这一句，make -C 表示调用执行的路径，也就是变量 KDIR，变量 KDIR 中有内核源码目录的路径。

PWD 表示当前目录。

modules 表示将驱动编译成模块的形式，也就是最终生成 KO 文件。

clean:

```
rm -rf *.o
```

在重新修改了源码之后，可以执行“make clean”命令来清除一些无用的中间文件，这里选择的是清除后缀为“o”的文件。

2.9 实验操作

2.9.1 内核目录的确认

用户必须要确认内核源码解压的目录，如果目录不正确，模块是无法编译的。

如下图所示，作者的内核源码目录是“/home/topeet/android4.0/iTop4412_Kernel_3.0”中。

```
root@ubuntu: ~  
root@ubuntu:~#  
root@ubuntu:~#  
root@ubuntu:~# ls /home/topeet/android4.0/iTop4412_Kernel_3.0  
arch          Documentation  lib           REPORTING-BUGS  
binary        drivers       MAINTAINERS  samples  
block         firmware     Makefile     scripts  
config_for_android fs           mm           security  
config_for_android_2M include      modem.patch  sound  
config_for_linux init         modules.builtin System.map  
config_for_ubuntu ipc          modules.order tools  
config_for_ubuntu_hdmi Kbuild      Module.symvers usr  
COPYING       Kconfig     net          virt  
CREDITS       kernel      pull_log.bat vmlinux  
crypto        kernel_readme.txt README       vmlinux.o  
root@ubuntu:~#
```

确认了内核目录，那么 Makefile 文件中就需要针对性修改引用的内核目录。如下图所示，脚本文件 Makefile 中变量 “KDIR” 的值就是

“/home/topeet/android4.0/iTop4412_Kernel_3.0”。

```
#!/bin/bash  
#通知编译器我们要编译模块的哪些源码  
#这里是编译itop4412_hello.c这个文件编译成中间文件mini_linux_module.o  
obj-m += mini_linux_module.o  
  
#源码目录变量，这里用户需要根据实际情况选择路径  
#作者是将Linux的源码拷贝到目录/home/topeet/android4.0下并解压的  
KDIR := /home/topeet/android4.0/iTop4412_Kernel_3.0  
  
#当前目录变量  
PWD ?= $(shell pwd)  
  
#make命名默认寻找第一个目标  
#make -C就是指调用执行的路径  
#$(KDIR) Linux源码目录，作者这里指的是/home/topeet/android4.0/iTop4412_Kernel_3.0  
#$(PWD) 当前目录变量  
#modules要执行的操作  
all:  
    make -C $(KDIR) M=$(PWD) modules  
  
#make clean执行的操作是删除后缀为o的文件  
clean:  
    rm -rf *.o
```

确认完之后，就可以进行下一步了。

2.9.2 内核以及文件系统的烧写

用户需要针对性的使用内核以及文件系统，教程是基于最小系统实现的。

首先 uboot 不用修改，也不用去烧写。

其次是内核，如下图所示，内核需要用到用户光盘目录 “iTOP-4412 精英版光盘资料\04_镜像_QT 文件系统\zImage” (如果是 SCP 的核心板则使用 SCP 目录下的，POP 的则使用 POP 目录下的)下的 “zImage” 。

最后是文件系统，如下图所示，使用的是最小文件系统，一个是启动文件 “ramdisk-uboot.img”，一个是最小文件系统 “system.img”，它们在 “视频 02-DriverModule_01\最小系统” 中。



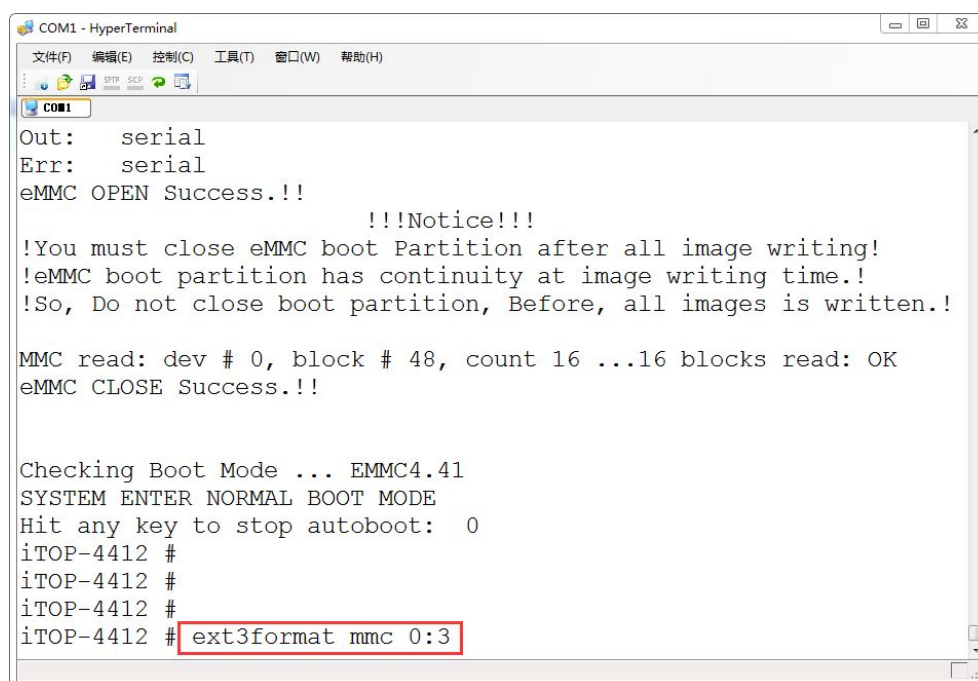
名称	修改日期	类型	大小
ramdisk-uboot.img	2015/7/3 ...	光盘映像...	637 KB
Read Me.txt	2015/8/3 ...	TXT 文件	1 KB
system.img	2014/10/...	光盘映像...	15,19...

2.9.3 烧写镜像

如果用户当前使用的是其它系统，那么则需要重新烧写一下。

下面介绍一下烧写过程。

在启动开发板之后，如下图所示，使用擦除命令 “ext3format mmc 0:3” 。



```
COM1 - HyperTerminal
文件(F)  编辑(E)  控制(C)  工具(T)  窗口(W)  帮助(H)

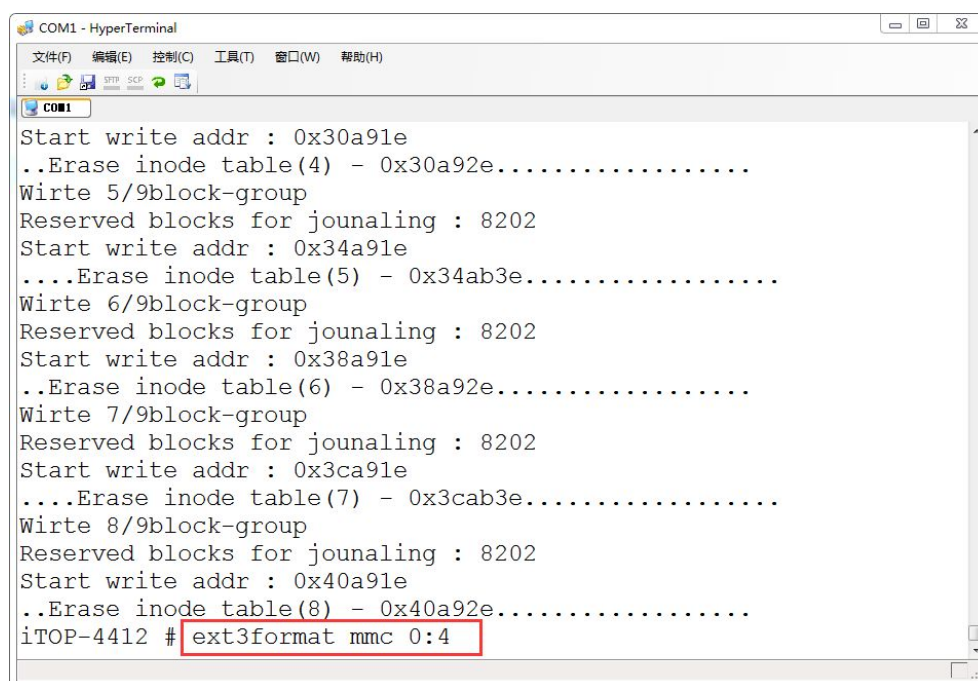
COM1
Out:  serial
Err:  serial
eMMC OPEN Success.!!

!!!Notice!!!
!You must close eMMC boot Partition after all image writing!
!eMMC boot partition has continuity at image writing time.!
!So, Do not close boot partition, Before, all images is written.!

MMC read: dev # 0, block # 48, count 16 ...16 blocks read: OK
eMMC CLOSE Success.!!

Checking Boot Mode ... EMMC4.41
SYSTEM ENTER NORMAL BOOT MODE
Hit any key to stop autoboot:  0
iTOP-4412 #
iTOP-4412 #
iTOP-4412 #
iTOP-4412 # ext3format mmc 0:3
```

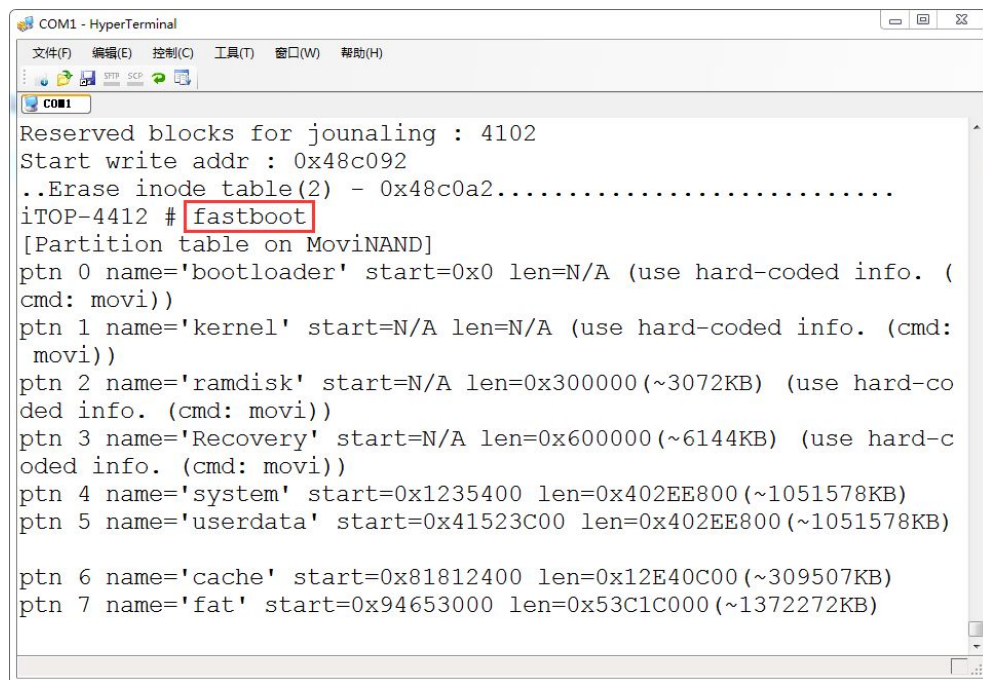
上图中的命令执行完毕之后，如下图所示，使用擦除命令“ext3format mmc 0:4”。



```
COM1 - HyperTerminal
文件(F)  编辑(E)  控制(C)  工具(T)  窗口(W)  帮助(H)

COM1
Start write addr : 0x30a91e
..Erase inode table(4) - 0x30a92e.....
Wirte 5/9block-group
Reserved blocks for jounaling : 8202
Start write addr : 0x34a91e
....Erase inode table(5) - 0x34ab3e.....
Wirte 6/9block-group
Reserved blocks for jounaling : 8202
Start write addr : 0x38a91e
..Erase inode table(6) - 0x38a92e.....
Wirte 7/9block-group
Reserved blocks for jounaling : 8202
Start write addr : 0x3ca91e
....Erase inode table(7) - 0x3cab3e.....
Wirte 8/9block-group
Reserved blocks for jounaling : 8202
Start write addr : 0x40a91e
..Erase inode table(8) - 0x40a92e.....
iTOP-4412 # ext3format mmc 0:4
```

上图中的命令执行完毕之后，如下图所示，执行“fastboot”命令，连接PC机器。这里需要将开发板的OTG接口和PC机的USB连接。



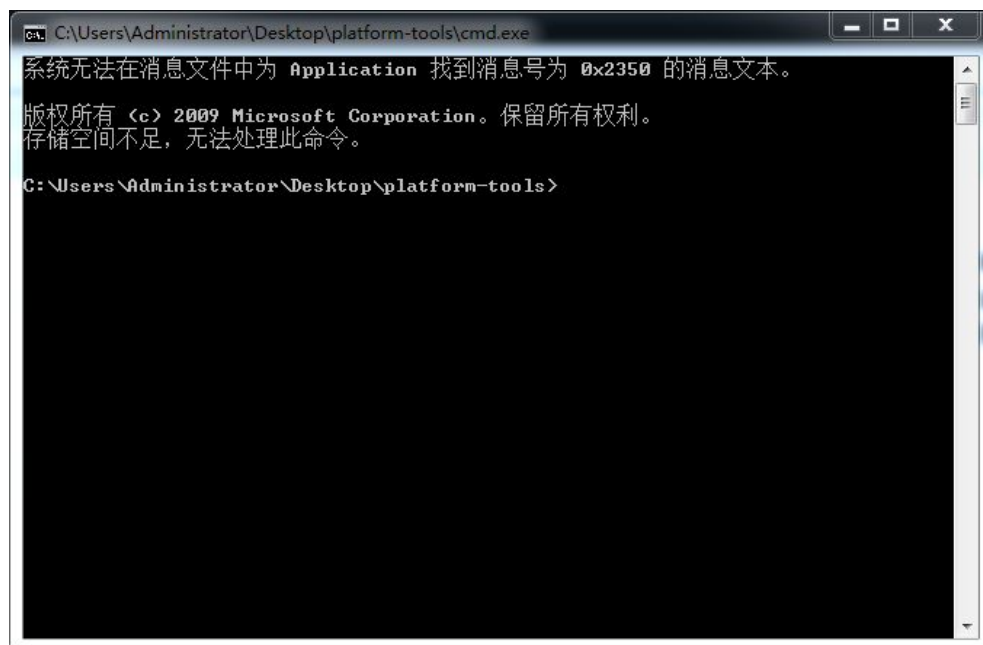
```
COM1 - HyperTerminal
文件(F)  编辑(E)  控制(C)  工具(T)  窗口(W)  帮助(H)

COM1
Reserved blocks for journaling : 4102
Start write addr : 0x48c092
..Erase inode table(2) - 0x48c0a2.....
iTOP-4412 # fastboot
[Partition table on MovinAND]
ptn 0 name='bootloader' start=0x0 len=N/A (use hard-coded info. (
cmd: movi))
ptn 1 name='kernel' start=N/A len=N/A (use hard-coded info. (cmd:
movi))
ptn 2 name='ramdisk' start=N/A len=0x300000(~3072KB) (use hard-co
ded info. (cmd: movi))
ptn 3 name='Recovery' start=N/A len=0x600000(~6144KB) (use hard-c
oded info. (cmd: movi))
ptn 4 name='system' start=0x1235400 len=0x402EE800(~1051578KB)
ptn 5 name='userdata' start=0x41523C00 len=0x402EE800(~1051578KB)

ptn 6 name='cache' start=0x81812400 len=0x12E40C00(~309507KB)
ptn 7 name='fat' start=0x94653000 len=0x53C1C000(~1372272KB)
```

将上一小节中介绍的内核文件以及文件系统等三个文件拷贝到烧写文件夹

“platform-tools” 中，打开 “cmd.exe” 程序，如下图所示。



```
C:\Users\Administrator\Desktop\platform-tools\cmd.exe
系统无法在消息文件中为 Application 找到消息号为 0x2350 的消息文本。
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。
存储空间不足，无法处理此命令。
C:\Users\Administrator\Desktop\platform-tools>
```

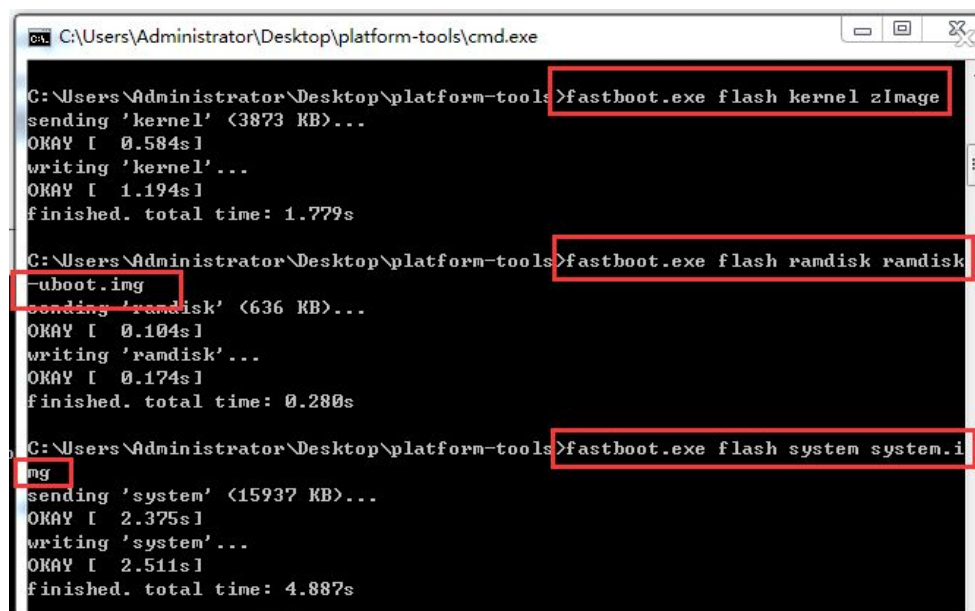
依次输入烧写命令

“fastboot.exe flash kernel zImage”

"fastboot.exe flash ramdisk ramdisk-uboot.img"

"fastboot.exe flash system system.img"

如下图所示。



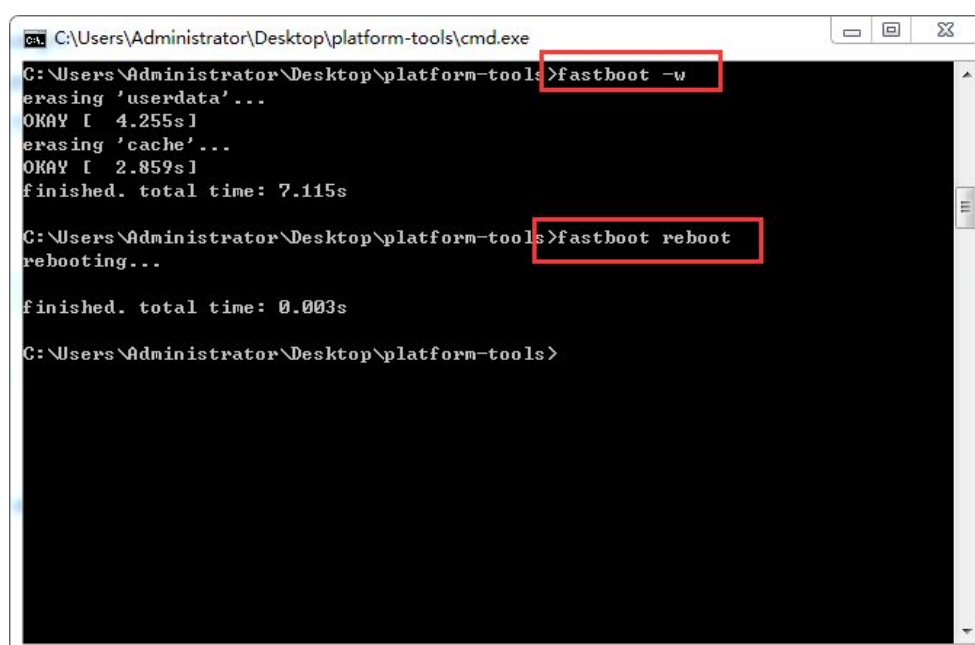
```
C:\Users\Administrator\Desktop\platform-tools\cmd.exe
C:\Users\Administrator\Desktop\platform-tools>fastboot.exe flash kernel zImage
sending 'kernel' (3873 KB)...
OKAY [ 0.584s]
writing 'kernel'...
OKAY [ 1.194s]
finished. total time: 1.779s

C:\Users\Administrator\Desktop\platform-tools>fastboot.exe flash ramdisk ramdisk-uboot.img
sending 'ramdisk' (636 KB)...
OKAY [ 0.104s]
writing 'ramdisk'...
OKAY [ 0.174s]
finished. total time: 0.280s

C:\Users\Administrator\Desktop\platform-tools>fastboot.exe flash system system.img
sending 'system' (15937 KB)...
OKAY [ 2.375s]
writing 'system'...
OKAY [ 2.511s]
finished. total time: 4.887s
```

烧写完成之后，输入擦除命令 "fastboot -w" 和重启命令 "fastboot reboot"，如下图

所示。



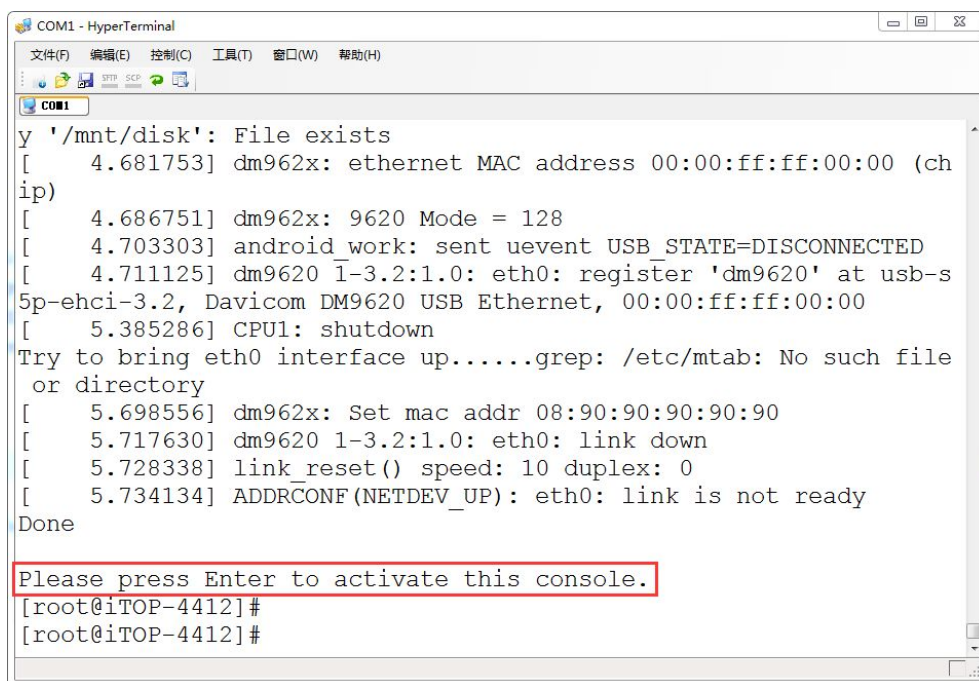
```
C:\Users\Administrator\Desktop\platform-tools\cmd.exe
C:\Users\Administrator\Desktop\platform-tools>fastboot -w
erasing 'userdata'...
OKAY [ 4.255s]
erasing 'cache'...
OKAY [ 2.859s]
finished. total time: 7.115s

C:\Users\Administrator\Desktop\platform-tools>fastboot reboot
rebooting...

finished. total time: 0.003s

C:\Users\Administrator\Desktop\platform-tools>
```

重启之后，超级终端会运行最小系统，如下图所示。

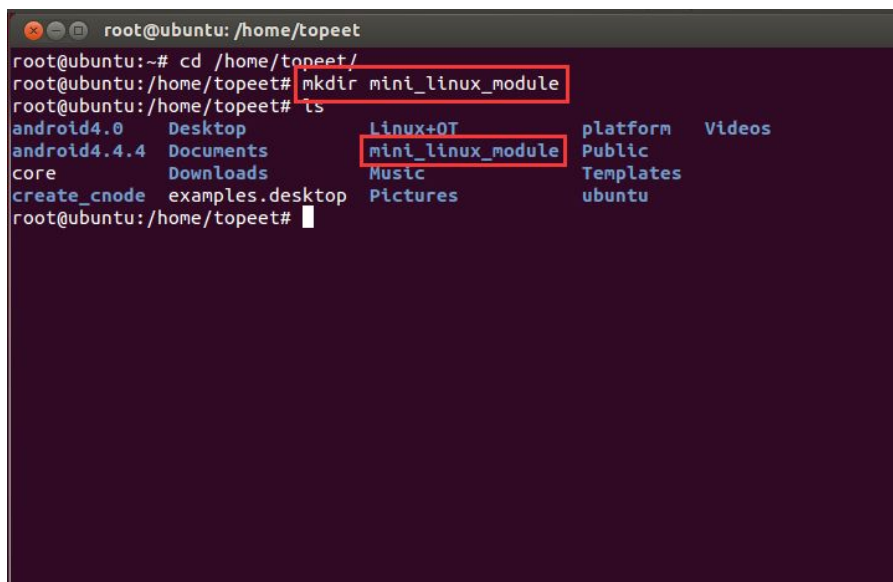


```
COM1 - HyperTerminal
文件(F) 编辑(E) 控制(C) 工具(T) 窗口(W) 帮助(H)
COM1
y '/mnt/disk': File exists
[ 4.681753] dm962x: ethernet MAC address 00:00:ff:ff:00:00 (chip)
[ 4.686751] dm962x: 9620 Mode = 128
[ 4.703303] android work: sent uevent USB_STATE=DISCONNECTED
[ 4.711125] dm9620 1-3.2:1.0: eth0: register 'dm9620' at usb-s5p-ehci-3.2, Davicom DM9620 USB Ethernet, 00:00:ff:ff:00:00
[ 5.385286] CPU1: shutdown
Try to bring eth0 interface up.....grep: /etc/mtab: No such file or directory
[ 5.698556] dm962x: Set mac addr 08:90:90:90:90:90
[ 5.717630] dm9620 1-3.2:1.0: eth0: link down
[ 5.728338] link_reset() speed: 10 duplex: 0
[ 5.734134] ADDRCONF(NETDEV_UP): eth0: link is not ready
Done
Please press Enter to activate this console.
[root@iTOP-4412]#
[root@iTOP-4412]#
```

到这一步，烧写文件系统的准备工作就完成了。

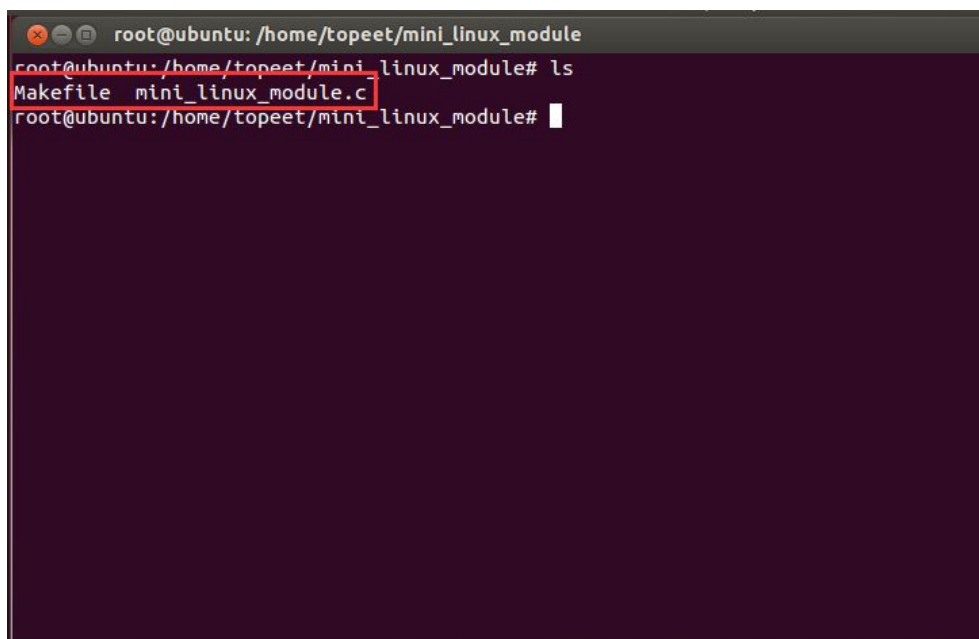
2.9.4 编译驱动模块

如下图所示，在“/home/topeet”目录下新建文件夹“mini_linux_module”。



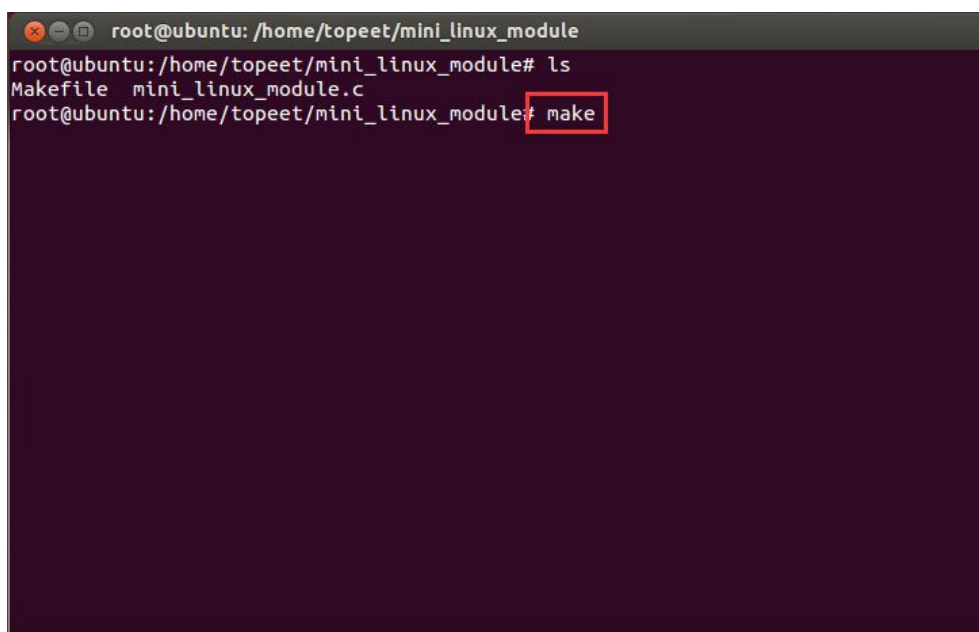
```
root@ubuntu: /home/topeet
root@ubuntu:~# cd /home/topeet/
root@ubuntu:/home/topeet# mkdir mini_linux_module
root@ubuntu:/home/topeet# ls
android4.0  Desktop  Linux+OT  platform  Videos
android4.4.4  Documents  mini_linux_module  Public
core  Downloads  Music  Templates
create_cnode  examples.desktop  Pictures  ubuntu
root@ubuntu:/home/topeet#
```

将文件拷贝前面编写好的 “mini_linux_module.c” 文件和对应的 “Makefile” 拷贝到该文件夹中，如下图所示。



```
root@ubuntu: /home/topeet/mini_linux_module
root@ubuntu: /home/topeet/mini_linux_module# ls
Makefile  mini_linux_module.c
root@ubuntu: /home/topeet/mini_linux_module#
```

如下图所示，在 “/home/topeet/mini_linux_module” 目录下执行命令 “make” 。



```
root@ubuntu: /home/topeet/mini_linux_module
root@ubuntu: /home/topeet/mini_linux_module# ls
Makefile  mini_linux_module.c
root@ubuntu: /home/topeet/mini_linux_module# make
```

如下图所示，有可能出现错误 “ *** missing separator” 。


```
root@ubuntu: /home/topeet/mini_linux_module
root@ubuntu: /home/topeet/mini_linux_module# ls
Makefile  mini_linux_module.c
root@ubuntu: /home/topeet/mini_linux_module# make
Makefile:23: *** missing separator (did you mean TAB instead of 8 spaces?).  Sto
p.
root@ubuntu: /home/topeet/mini_linux_module#
```

如果编译不通过，打开文件 Makefile，如下图所示。

```
root@ubuntu: /home/topeet/test
#!/bin/bash
#通知编译器我们要编译模块的哪些源码
#编译itop4412_hello.c这个文件编译成中间文件mini_linux_module.o
obj-m += mini_linux_module.o

#源码目录变量，这里用户需要根据实际情况选择路径
#作者是将Linux的源码拷贝到目录/home/topeet/android4.0下并解压的
KDIR := /home/topeet/android4.0/iTop4412_Kernel_3.0

#当前目录变量
PWD ?= $(shell pwd)

#make命名默认寻找第一个目标
#make -C就是指调用执行的路径
#$(KDIR)Linux源码目录，作者这里指的是/home/topeet/android4.0/iTop4412_Kernel_3.0
#$(PWD)当前目录变量
modules要执行的操作
all:
    make -C $(KDIR) M=$(PWD) modules

#make clean执行的操作是删除后缀为o的文件
clean:
    rm -rf *.o

"Makefile" [dos] 23L, 804C                                17,1                                All
```

如下图所示，在 Ubuntu 下使用 Vim 编辑器将红色部分处理一下，删除红色部分代码的空格部分，再输入“Tab”。

```
root@ubuntu: /home/topeet/mini_linux_module
#!/bin/bash
#通知编译器我们要编译模块的哪些源码
#这里是编译itop4412_hello.c这个文件编译成中间文件mini_linux_module.o
obj-m += mini_linux_module.o

#源码目录变量，这里用户需要根据实际情况选择路径
#作者是将Linux的源码拷贝到目录/home/topeet/android4.0下并解压的
KDIR := /home/topeet/android4.0/iTop4412_Kernel_3.0

#当前目录变量
PWD ?= $(shell pwd)

#make命名默认寻找第一个目标
#make -c就是指调用执行的路径
#$(KDIR)Linux源码目录，作者这里指的是/home/topeet/android4.0/iTop4412_Kernel_3.0
#$(PWD)当前目录变量
#modules要执行的操作
all:
    make -C $(KDIR) M=$(PWD) modules

#make clean执行的操作是删除后缀为o的文件
clean:
    rm -rf *.o

-- INSERT --
```

保存退出，然后再执行编译命令“make”，如下图所示，编译成功。

```
root@ubuntu: /home/topeet/mini_linux_module
root@ubuntu:/home/topeet/mini_linux_module# ls
Makefile mini_linux_module.c
root@ubuntu:/home/topeet/mini_linux_module# make
Makefile:23: *** missing separator (did you mean TAB instead of 8 spaces?). Stop.
root@ubuntu:/home/topeet/mini_linux_module# vim Makefile
root@ubuntu:/home/topeet/mini_linux_module# make
make -C /home/topeet/android4.0/iTop4412_Kernel_3.0 M=/home/topeet/mini_linux_module modules
make[1]: Entering directory `/home/topeet/android4.0/iTop4412_Kernel_3.0'
CC [M] /home/topeet/mini_linux_module/mini_linux_module.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/topeet/mini_linux_module/mini_linux_module.mod.o
LD [M] /home/topeet/mini_linux_module/mini_linux_module.ko
make[1]: Leaving directory `/home/topeet/android4.0/iTop4412_Kernel_3.0'
root@ubuntu:/home/topeet/mini_linux_module#
```

如下图所示，使用查看命令“ls”，可以看到生成了驱动模块文件“mini_linux_module.ko”，这个文件就是最后要加载到开发板中的文件。


```
root@ubuntu: /home/topeet/mini_linux_module
root@ubuntu:/home/topeet/mini_linux_module# ls
Makefile mini_linux_module.c
root@ubuntu:/home/topeet/mini_linux_module# make
Makefile:23: *** missing separator (did you mean TAB instead of 8 spaces?). Stop.
root@ubuntu:/home/topeet/mini_linux_module# vim Makefile
root@ubuntu:/home/topeet/mini_linux_module# make
make -C /home/topeet/android4.0/iTop4412_Kernel_3.0 M=/home/topeet/mini_linux_module modules
make[1]: Entering directory `/home/topeet/android4.0/iTop4412_Kernel_3.0'
CC [M] /home/topeet/mini_linux_module/mini_linux_module.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/topeet/mini_linux_module/mini_linux_module.mod.o
LD [M] /home/topeet/mini_linux_module/mini_linux_module.ko
make[1]: Leaving directory `/home/topeet/android4.0/iTop4412_Kernel_3.0'
root@ubuntu:/home/topeet/mini_linux_module# ls
Makefile mini_linux_module.mod.c Modules.order
mini_linux_module.c mini_linux_module.mod.o Module.symvers
mini_linux_module.ko mini_linux_module.o
root@ubuntu:/home/topeet/mini_linux_module#
```

2.9.5 加载驱动

将 mini_linux_module.ko 驱动模块文件拷贝到 U 盘或者 TF，然后接到开发板上。

将 U 盘或者 TF 卡加载，可以参考使用手册 11.3.3 qt 挂载盘符。

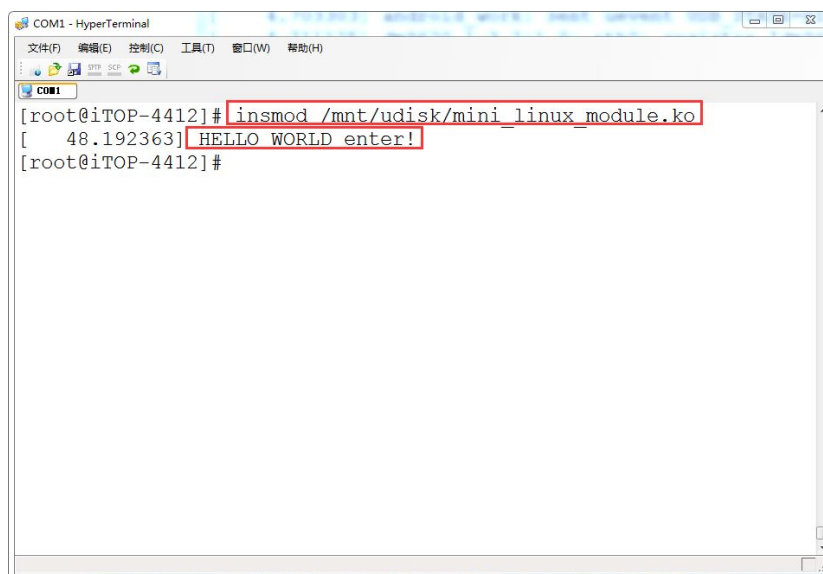
作者这里使用的是 U 盘，挂载之后需要的加载模块驱动 KO 文件，在 mnt/udisk 下。如下

图所示。

```
COM1 - HyperTerminal
文件(F) 编辑(E) 控制(C) 工具(T) 窗口(W) 帮助(H)
COM1
[ 2274.143776] usb 1-3.3: Product: Cruiser Blade
[ 2274.148011] usb 1-3.3: Manufacturer: SanDisk
[ 2274.152262] usb 1-3.3: SerialNumber: 20051536210C7A80EAD
[ 2274.165655] scsi0 : usb-storage 1-3.3:1.0
[ 2275.171472] scsi 0:0:0:0: Direct-Access SanDisk Cruiser B
ade 1.26 PQ: 0 ANSI: 5
[ 2275.182758] sd 0:0:0:0: [sda] 7821312 512-byte logical blocks:
(4.00 GB/3.72 GiB)
[ 2275.186893] sd 0:0:0:0: Attached scsi generic sg0 type 0
[ 2275.195695] sd 0:0:0:0: [sda] Write Protect is off
[ 2275.199937] sd 0:0:0:0: [sda] Write cache: disabled, read cache:
enabled, doesn't support DPO or FUA
[ 2275.219122] sda: sda1
[ 2275.224217] sd 0:0:0:0: [sda] Attached SCSI removable disk

[root@iTOP-4412]#
[root@iTOP-4412]# mkdir /mnt/udisk
[root@iTOP-4412]# mount /dev/sda1 /mnt/udisk/
[root@iTOP-4412]#
```

然后使用命令 “insmod /mnt/udisk/mini_linux_module.ko” 加载模块，如下图所示。



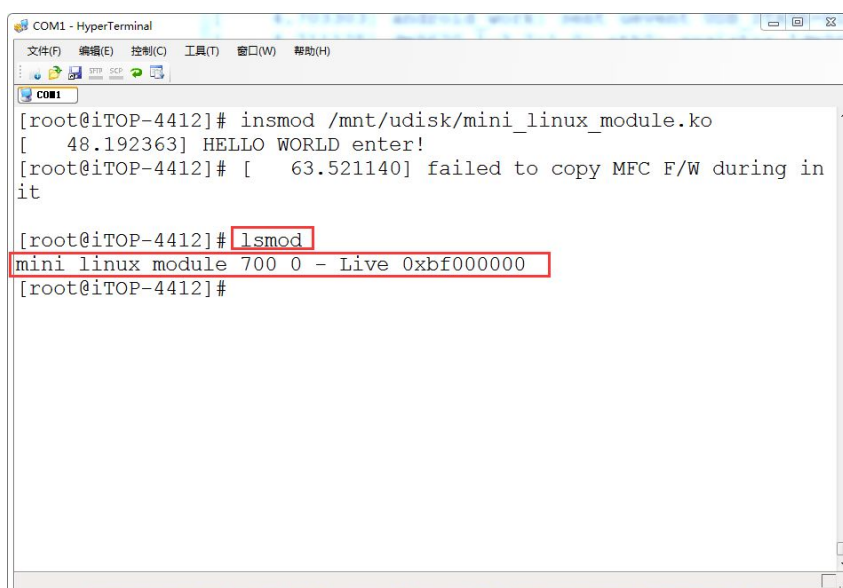
```
COM1 - HyperTerminal
文件(F)  编辑(E)  控制(C)  工具(T)  窗口(W)  帮助(H)

[root@iTOP-4412]# insmod /mnt/udisk/mini_linux_module.ko
[ 48.192363] HELLO WORLD enter!
[root@iTOP-4412]#
```

如上图所示，可以看到加载过程中打印了信息 “Hello World enter!”，说明驱动已经加载到 Linux 中去了。

2.9.6 卸载驱动

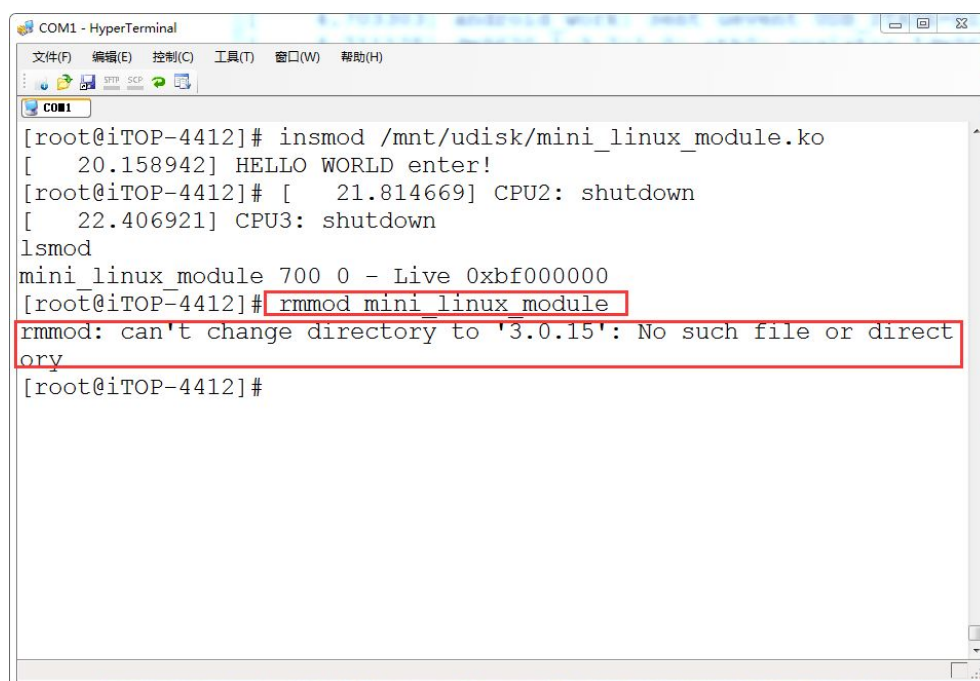
卸载驱动前面，先使用命令 “lsmod”，查看一下模块信息，如下图所示。



```
COM1 - HyperTerminal
文件(F)  编辑(E)  控制(C)  工具(T)  窗口(W)  帮助(H)

[root@iTOP-4412]# insmod /mnt/udisk/mini_linux_module.ko
[ 48.192363] HELLO WORLD enter!
[ 63.521140] failed to copy MFC F/W during in
it
[root@iTOP-4412]# lsmod
mini linux module 700 0 - Live 0xbf000000
[root@iTOP-4412]#
```

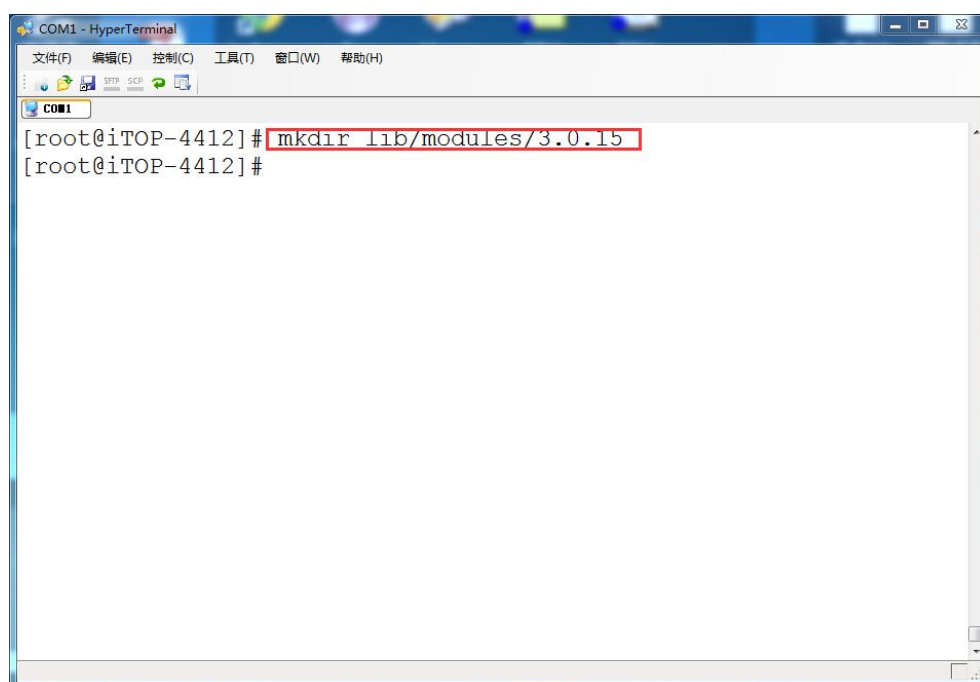
如下图所示，使用命令 “rmmod mini_linux_module” 卸载驱动模块。



```
COM1 - HyperTerminal
文件(F)  编辑(E)  控制(C)  工具(T)  窗口(W)  帮助(H)

COM1
[root@iTOP-4412]# insmod /mnt/udisk/mini_linux_module.ko
[ 20.158942] HELLO WORLD enter!
[root@iTOP-4412]# [ 21.814669] CPU2: shutdown
[ 22.406921] CPU3: shutdown
lsmod
mini_linux_module 700 0 - Live 0xbf000000
[root@iTOP-4412]# rmmod mini_linux_module
rmmod: can't change directory to '3.0.15': No such file or directory
[root@iTOP-4412]#
```

如上图所示，会报错无法卸载，提示没有文件夹。这里根据提示，使用命令 “mkdir lib/modules/3.0.15” 新建目录 “lib/modules/3.0.15”，如下图所示。

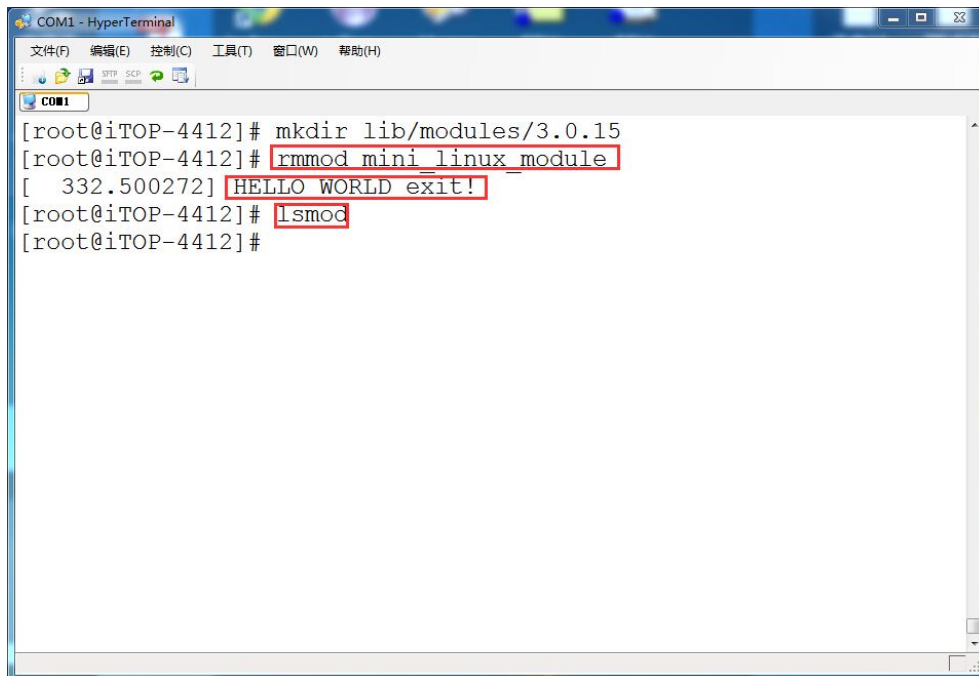


```
COM1 - HyperTerminal
文件(F)  编辑(E)  控制(C)  工具(T)  窗口(W)  帮助(H)

COM1
[root@iTOP-4412]# mkdir lib/modules/3.0.15
[root@iTOP-4412]#
```

新建文件夹之后，再使用卸载驱动模块的命令“rmmod mini_linux_module”，如下图所示，可以看到打印出了在卸载驱动函数里面添加的打印信息：Hello world exit！

最后使用命令：lsmod，对比前面的lsmod，发现已经没有了加载的模块驱动了。



```
COM1 - HyperTerminal
文件(F)  编辑(E)  控制(C)  工具(T)  窗口(W)  帮助(H)

COM1
[ root@iTOP-4412 ] # mkdir lib/modules/3.0.15
[ root@iTOP-4412 ] # rmmod mini_linux_module
[ 332.500272 ] HELLO WORLD exit!
[ root@iTOP-4412 ] # lsmod
[ root@iTOP-4412 ] #
```