linux内核模块化编程

快速编写自己的功能代码，然后以模块的形式添加到Linux操作系统中，然后测试，发现不行，卸载模块，继续修改代码

linux内核模块主要由以下几个部分组成：

- 模块加载函数（必须）：当通过insmod命令加载内核模块时，模块的加载函数会自动被内核执行，完成本模块相关初始化工作；
- 模块卸载函数（必须）：当通过rmmod命令卸载模块时，模块的卸载函数会自动被内核执行，完成与模块加载函数相反的功能；
- 模块许可证声明（必须）：模块许可证（LICENCE）声明描述内核模块的许可权限，如果不声明LICENCE,模块被加载时将收到内核被污染的警告。大多数
- 模块参数（可选）：模块参数是模块被加载的时候可以被传递给他的值，它本身对应模块内部的全局变量；
- 模块导出符号（可选）：内核模块可以导出符号(symbol,对应于函数或变量)，这样其他模块可以使用本模块中的变量或函数；
  模块作者等信息声明（可选）

如何进行模块化编程

- `printf` 主要用于用户空间程序，通过标准输出输出到终端。
- `printk` 主要用于内核空间，可以在内核代码中直接调用，用于调试和记录信息。

- 当你想查看系统引导过程中的日志信息时，可以使用以下命令：

```
dmesg | less
```

- 可以使用 `grep` 命令来筛选出特定的文件系统类型（比如 revofs）的输出。下面是一个示例命令：

```
dmesg | grep revofs
```

内核模块化编程步骤

1. 包含linux/init.h和inux/module.h这两个头文件；

   通过MODULE LICENSE("GPL"),告诉内核你的模块遵从"GPL"协议。Linux能够成功一个关键因素就是遵循了GPL

   MODULE_AUTHOR("yikoulinux")用来指定编写这个模块的作者，可以不写。

2. 开始编写功能代码，注意这里无main函数，而是模块的入口函数

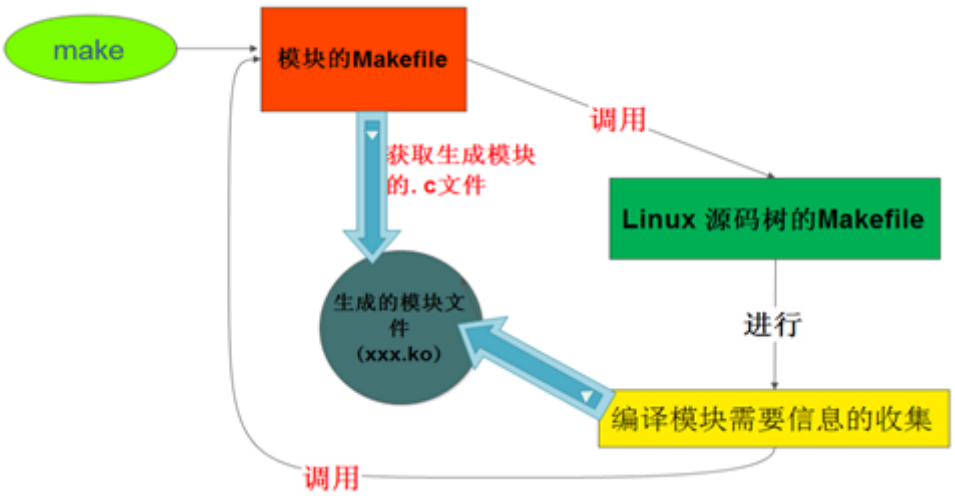
   在模块加载到Linux内核的时候，Linux内核会调用这个函数模块的函数....

3. 告诉内核，你的模块入入口和模块出口。Linux内核提供了两个宏，分别是：module_init 和 module_exit.

printk

printk的用法和printf类似，`print`用于用户空间，`printk`用于内核空间。用printk函数时，内核会根据日志级别，可能把消息打印到当前控制台上，这个控制台通常是一个字符模式的终端、一个串口打印机或是一个并口打印机。

模块的编译



模块的编译分两步：

第一步：调用linux源码树的Makefile进行收集编译一个模块所需要的信息

第二步：linux源码树的Makefile在收集完信息后，调用模块的Makefile。获取需要编译成模块的“.c”文件，最后生成模块文件

makefile编写

```
KDIR      :制定当前Linux操作系统源代码路径，即编译生成的模块是在当前系统中使用
```

```
PWD:=$(shell pwd)获得当前路径
```

```
ifneq(S(KERNELRELEASE),)
$(info "2nd")
obj-m:=hello.o
else
#在使用时kdir要修改为你自己的内核的顶级目录
KDIR :=/lib/modules/$(shell uname -r)/build
PND :=$(shell pwd)
all:
    $(info "1st")
    make -C $(KDIR) M=$(PWD) modules

#-C 选项的作用是指将当前工作目录转移到你所指定的位置。
#“M=”选项的作用是，当用户需要以某个内核为基础编译一个外部模块的话，
#需要在make modules 命令中加入“M=dir”，
#程序会自动到你所指定的dir目录中查找模块源码，将其编译，生成KO文件。
clean:
    rm -f *.ko *.o *.mod.o *.symvers *.cmd *.mod.c *.order
endif


#使用make命令编译模块，由于makefile文件无法正确的处理带空格的路径名，
#确保路径没有空格符
#该命令是make modules命令的扩展，-C选项的作用是指将当前的工作目录转移到制定的 目录
#即（KDIR）目录，程序到（shellpwd）当前目录查找模块源码，将其编译，生成.ko文件
```

执行过程：

- 在模块的源代码目录下执行make，此时，宏“KERNELRELEASE”【内核源码树的Makefile会定义】没有定义，因此进入else
- 记录内核路径KDIR和当前工作目录PWD；
- 由于make 后面没有目标，所以make会在Makefile中的第一个不是以.开头的目标作为默认的目标执行，于是all成为make的目标；

  all：的第一个命令 $(info "lst")打印提示信息
- make的第二条命令会执行 make -C $(KDIR) M=$(PWD) modules

```
make -C /lib/modules/3.2.0-29-generic-pae/build M=/home/peng/driver/1/module
modules
```

-C 表示到存放内核的目录执行其Makefile，

M=$(PWD)表示返回到当前目录，

modules表示编译成模块的意思

- 这么写是由于内核源码树的顶层Makefile告诉我们的

```
# External module support.
# When building external modules the kernel used as basis is considered
# read-only, and no consistency checks are made and the make
# system is not used on the basis kernel. If updates are required
# in the basis kernel ordinary make commands (without M=...) must
# be used.
#
# The following are the only valid targets when building external
# modules.
# make M=dir clean     Delete all automatically generated files
# make M=dir modules   Make all modules in specified dir
# make M=dir           Same as 'make M=dir modules'
# make M=dir modules_install
#                      Install the modules built in the module directory
#                      Assumes install directory is already created

# We are always building modules
#
modules: $(module-dirs)
    @$(kecho) '  Building modules, stage 2.';
    $(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modpost
```

- 找到modules目标后，接下来Linux源码树的顶层Makeflle就需要知道是将哪些".c"文件编译成模块

  模块的Makefile文件会告诉它，接下来回调模块的makefile，此时KERNELRELEASE已经在Linux内核源码树的顶层Makefile中定义过了，所以此时它获得信息是：

```
obj-m:=hello.o
```

  - obj-m表示会将hello.o目标编译成.ko模块；它告诉linux源码树顶层Makefile是动态编译（编译成模块）
  - linux源码树顶层Makefile会根据hello.o找到hello.c文件

- 将模块文件hello.c编译为.o,然后再将多个目标链接为.ko

```
root@ubuntu:/home/peng/driver/1/module# make
"1st"
make -C /lib/modules/3.2.0-29-generic-pae/build M=/home/peng/driver/1/module modules
make[1]: 正在进入目录 `/usr/src/linux-headers-3.2.0-29-generic-pae'  进入内核源码树
"2nd"
  CC [M]  /home/peng/driver/1/module/hello.o     产生.o文件
  Building modules, stage 2.
"2nd"
  MODPOST 1 modules
  CC      /home/peng/driver/1/module/hello.mod.o
  LD [M]  /home/peng/driver/1/module/hello.ko    链接生成.ko模块文件
make[1]:正在离开目录 `/usr/src/linux-headers-3.2.0-29-generic-pae'
```

由执行结果可知，Makefile最终被调用了三次

1） 执行命令make调用

2） 被linux内核源码树的顶层Makefile调用，产生.o文件

3） 被linux源码树顶层Makefile调用，将.o文件链接生成.ko文件

模块的加载、卸载

1. 模块的加载命令

```
insmod xxx.ko
```

例如：在ubuntu系统中添加自己写的模块

sudo insmod hello.ko

注意：在Linux系统中只有超级用户权限才可以添加模块到内核

2. 查看系统中的模块命令

```
lsmod
```

例如：在系统中搜索自己添加的hello模块

```
sudo lsmod | grep hello
```

3. 卸载模块命令

```
sudo rmmod 模块名
```

例如：卸载系统中的hello模块

```
sudo mmod hello
```

4. 查看加载模块和卸载模块通过printk打印的信息命令

```
dmesg或dmesg | tail
```

这个命令主要是从Linux内核的ring buffer(环形缓冲区）中读取信息的。

那什么是ring buffer呢?

在Linux系统中，所有通过printk打印出来的信息都会送到ring buffer中。我们知道，我们打印出来的信息是需要在控制台设备上显示的。在Linux内核初始化的时候，控制台设备并没有初始化的时候，使用printk会不会有问题

控制台设备，因为此时printk只是把信息输送到ring buffer中，等控制台设备初始化好后，在根据ring buffer中消息的优先级决定是否需要输送到控制台设备上。

如何清空ring buffer呢?

```
 sudo dmesg -c
```