module的加载和卸载

**一、背景介绍**

linux是个宏内核，为了方便的将功能增加进内核，linux采用module模式。

程序员创建实现各个功能的module，创建完成之后，生成ko后缀的二进制文件。当需要使用的时候，使用模块加载命令将module加载进正在运行的内核，当不需要此功能的时候，可以使用module卸载命令将此模块从正在运行的模块中卸载。

**二、知识点：**

1.两个重要的函数：init和exit

使用名称类似init\_XXX的函数作为模块被加载时调用的函数。

为了特别标记，实现此函数时使用空宏\_\_init注释，仅标记此函数为init函数。

使用名称类似exit\_XXX的函数作为模块被卸载时调用的函数。

为了特别标记，实现此函数时使用空宏\_\_exit注释，仅标记此函数为exit函数。

因为这两个函数不被其他函数调用，只做模块初始化和卸载时使用，所以通常声明为static函数。

2.两个更重要的宏：module\_init和module\_exit

module\_init(init\_XXX); 可以将函数init\_XXX声明为模块初始化调用。

module\_exit(exit\_XXX); 可以将函数exit\_XXX声明为模块卸载时调用。

3.编译模块的makefile

有两种方法将模块加载到内核，一种是编译内核时一起编译进内核。另一种是编译为ko文件，然后在需要时加载进内核或从内核卸载。

由内核根目录下的.config文件决定模块是以何种方式编译。.config文件由各级目录下的Kconfig文件、Makefile文件共同管理，由make menuconfig命令进行配置。

因为编译进内核和编译成模块两种方法可供选择，所以编译模块的makefile文件主体结构是个if语句，根据KERNELRELEASE关键字是否为空来判断是否是内核编译。

若KERNELRELEASE宏不为空，则是编译进内核，只需要设定obj-m对应的对象文件即可；

若KERNELRELEASE为空，则是编译为模块，除了设定obj-m之外，还要定义KERNELDIR（指向内核源码树）和PWD（等于当前目录）两个变量，然后添加目标：依赖和规则，规则是 $(MAKE) –C $(KERNELDIR) M=$(PWD) modules

完整的makefile如下：

ifneq ($(KERNELRELEASE),)

obj-m := scull.o

else

obj-m:=scull.o

module-objs:=module.o

KERNELDIR?=/lib/modules/$(shell uname -r)/build

PWD:=$(shell pwd)

default:

$(MAKE) -C ${KERNELDIR} M=${PWD} modules

endif

.PHNOY:clean

clean:

rm -rf \*.o \*.ko \*.mod.\* modules.\* Module.\*

4.模块的加载

insmod moduleName.ko

5.模块的卸载

rmmod moduleName.ko

**三、示例演示**

/\*tempM.c\*/

#include <linux/init.h>

#include <linux/module.h>

MODULE\_LICENSE("Dual BSD/GPL");

static int \_\_init init\_temp(void)

{

return 0;

}

static void \_\_exit exit\_temp(void)

{

}

module\_init(init\_temp);

module\_exit(exit\_temp);

/\*Makefile\*/

ifneq ($(KERNELRELEASE),)

obj-m:=tempM.o

else

obj-m:=tempN.o

KERNELDIR?=/lib/modules/$(shell uname -r)/build

PWD:=$(shell pwd)

default:all

all:

$(MAKE) -C $(KERNELDIR) M=$(PWD) modules

endif

使用下面命令编译并测试

$ make

$ insmod tempM.ko

$ rmmod tempM.ko