# Linux内核模块基础

# 1内核简单模块的编写

通过命令date可以获取当前系统时间，如下面示例。

```
linux:~/programming # date
Thu Nov 11 05:03:01 EST 2010
linux:~/programming # date '+%s'
1289469658
linux:~/programming #
```

下面我们通过编写一个简单的内核模块直接获取当前系统时间。

## 1.1模块源码编写

在Linux内核源码中，定义了一个struct timeval结构体，结构体中有两个成员变量tv_sec，tv_usec，分别保存当前系统时间的秒和毫秒，time_t和suseconds_t类型变量在x86架构中，均为long型，变量类型定义在文件include/linux/time.h中。

```
00018: struct timeval {
00019: time_t tv_sec; /* seconds */
00020: suseconds_t tv_usec; /* microseconds */
00021: };
00022:
```

模块源码如下：

```
00001:
00002: #include <linux/ module.h>
00003: #include <linux/ time.h>
00004:
00005: static char modname[] = "time";
00006:
00007: extern struct timespec xtime;
00008:
00009: int init_module( void )
00010: {
00011: struct timeval tv;
00012: printk( "Installing %s module.", modname );
00013: do_gettimeofday(&tv);
00014: printk("\njiffies:%lu, tv.tv_sec:%lu, tv.tv_nsec:%lu ", jiffies, tv.tv_sec, tv.tv_usec);
```

```
00015:
00016: return 0;
00017: }
00018:
00019:
00020: void cleanup_module( void )
00021: {
00022: printk( "\nRemoving %s module.", modname );
00023: }
00024:
00025: MODULE_LICENSE("GPL");
00026:
```

1.2Makefile

创建一个Makefile，执行**make**，即可编译生成内核模块，生成后缀名为.ko的文件。

```
linux:~/programming # ls
Makefile  time.c
linux:~/programming # make
make -C /lib/modules/2.6.32.12-0.7-default/build SUBDIRS=/root/programming modules
make[1]: Entering directory `/usr/src/linux-2.6.32.12-0.7-obj/x86_64/default'
make -C ../../../linux-2.6.32.12-0.7 O=/usr/src/linux-2.6.32.12-0.7-obj/x86_64/default/.
modules
  CC [M]  /root/programming/time.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/programming/time.mod.o
  LD [M]  /root/programming/time.ko
make[1]: Leaving directory `/usr/src/linux-2.6.32.12-0.7-obj/x86_64/default'
rm -r .tmp_versions *.mod.c .*.cmd *.o
linux:~/programming #
```

Makefile内容：

```
ifneq   ($(KERNELRELEASE),)
obj-m   := time.o

else
KDIR   := /lib/modules/$(shell uname -r)/build
PWD   := $(shell pwd)
default:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
    rm -r .tmp_versions *.mod.c .*.cmd *.o
endif
```

注意：在default：后面的$（MAKE） … … 和rm –r … …两行前面必须是Tab键，不能为空格或其他字符，否则执行make时，会报告“Makefile:10: *** missing separator. Stop.”错误。

### 1.3模块加载

执行make，编译生成模块.ko文件后，就可以通过**insmod**命令来加载模块。

```
linux:~/programming # insmod time.ko
linux:~/programming #
```

通过**lsmod**命令可以查看驱动是否成功加载到内核中。

```
linux:~/programming # lsmod
Module                  Size  Used by
time                    1026  0
raw                     3772  0
bridge                  59850  1
stp                     2147  1 bridge
llc                     5880  2 bridge,stp
rdma_ucm                11920  0
rdma_cm                 30822  1 rdma_ucm
iw_cm                   9522  1 rdma_cm
```

通过insmod命令加载刚编译成功的time.ko模块后，似乎系统没有反应，也没看到打印信息。而事实上，内核模块的打印信息一般不会打印在终端上。驱动的打印都在内核日志中，我们可以使用**dmesg**命令查看内核日志信息。

```
linux:~/programming # dmesg |tail
[71912.919422] Installing time module.
[71912.919426] jiffies:4312900540, tv.tv_sec:1289489871, tv.tv_nsec:977292
[72129.803130] Removing time module.
linux:~/programming # date '+%s'
1289489871
linux:~/programming #
```

内核模块time.ko获取到的当前系统时间为1289489871秒，与执行date '+%s'命令获取到的值一致。

## 2内核模块版本与符号表

在编写和使用内核模块过程中，会发现在某个内核版本上编译的模块只能在当前内核版本中使用。若模块版本号与当前内核版本号不匹配导致就会无法加载，提示“*insmod: error inserting 'time.ko': -1 Invalid module format*”，内核会打印类似信息“*time: version magic '2.6.32.12-0.7-default SMP mod_unload modversions ' should be '2.6.18-92.el5 SMP mod_unload gcc-4.1*'”。

```
[root@linux_driver ~]# insmod time.ko
insmod: error inserting 'time.ko': -1 Invalid module format
[root@linux_driver ~]#
```

### 2.1内核模块版本号

查看内核模块版本信息的命令为modinfo，如查看刚才我们编写的time.ko。

```
linux:~/programming # modinfo time.ko
filename:       time.ko
license:        GPL
srcversion:     9642FEB2E02F9232C0E72CE
depends:
vermagic:       2.6.32.12-0.7-default SMP mod_unload modversions
linux:~/programming #
```

模块的版本号在“vermagic“一项，当前系统中使用的模块版本号都是相同的。

模块版本号是哪里决定的？我们是否可以更改？我们是否可以在当前系统中，编译其他内核版本的模块？

#### 2.1.1模块版本号的确定

在make编译模块时，通过-C参数制定内核源码头文件位置。前面我们编译time模块内核源码头文件位置为/lib/modules/2.6.32.12-0.7-default/build。

*make -C /lib/modules/2.6.32.12-0.7-default/build SUBDIRS=/root/programming modules*

```
linux:~/programming # ls -l /lib/modules/2.6.32.12-0.7-default/build
lrwxrwxrwx 1 root root 47 Nov  5 08:17 /lib/modules/2.6.32.12-0.7-default/build ->
/usr/src/linux-2.6.32.12-0.7-obj/x86_64/default
linux:~/programming # cd /lib/modules/2.6.32.12-0.7-default/build
linux:/lib/modules/2.6.32.12-0.7-default/build # ls
.config Makefile Module.base Module.supported Module.symvers include include2 scripts
linux:/lib/modules/2.6.32.12-0.7-default/build # du -sh
7.5M
linux:/lib/modules/2.6.32.12-0.7-default/build #
```

我们来分析模块版本号的确定

**vermagic: 2.6.32.12-0.7-default SMP mod_unload modversions**

在RHEL5系统中，模块版本号vermagic由include/linux/vermagic.h和include/linux/utsrelease.h两个文件的内容来决定，即vermagic就为VERMAGIC_STRING。

文件include/linux/utsrelease.h的内容如下：

```
#define UTS_RELEASE "2.6.18-92.el5"
```

文件include/linux/vermagic.h的内容如下：

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

/* Simply sanity version stamp for modules. */
#ifdef CONFIG_SMP
#define MODULE_VERMAGIC_SMP "SMP "
#else
#define MODULE_VERMAGIC_SMP ""
#endif
#ifdef CONFIG_PREEMPT
#define MODULE_VERMAGIC_PREEMPT "preempt "
#else
#define MODULE_VERMAGIC_PREEMPT ""
#endif
#ifdef CONFIG_MODULE_UNLOAD
#define MODULE_VERMAGIC_MODULE_UNLOAD "mod_unload "
#else
#define MODULE_VERMAGIC_MODULE_UNLOAD ""
#endif
#ifndef MODULE_ARCH_VERMAGIC
#define MODULE_ARCH_VERMAGIC ""
#endif

#define VERMAGIC_STRING                                 \
        UTS_RELEASE " "                                 \
        MODULE_VERMAGIC_SMP MODULE_VERMAGIC_PREEMPT             \
        MODULE_VERMAGIC_MODULE_UNLOAD MODULE_ARCH_VERMAGIC          \
        "gcc-" __stringify(__GNUC__) "." __stringify(__GNUC_MINOR__)
```

而在SLES11.1内核2.6.32.12-0.7-default的模块版本号VERMAGIC_STRING由scripts/mod/modpost可执行文件确定。

```
linux:/lib/modules/2.6.32.12-0.7-default/build # grep "VERMAGIC_STRING" ./* -r
Binary file ./scripts/mod/modpost matches
linux:/lib/modules/2.6.32.12-0.7-default/build #
```

### 2.1.2模块版本号的修改

前面分析了模块版本号由VERMAGIC_STRING确定产生，若我们需要修改模块版本号或希望在当前内核版本中编译其他内核的模块（注意gcc大版本和CPU架构i686/x86_64保持一致），只需要修改控制模块的VERMAGIC_STRING值即可。

如我们将示例中的time模块在RHEL5.2内核中编译RHEL5.3内核模块，然后在RHEL5.3系统上可以加载。

```
[root@linux_driver linux]# cat utsrelease.h
#define UTS_RELEASE "2.6.18-92.el5"
[root@linux_driver linux]#
[root@linux_driver linux]# cat utsrelease.h
#define UTS_RELEASE "2.6.18-194.el5"
[root@linux_driver linux]#
```

### 2.1.3编译非当前内核版本模块

在模块版本号的修改介绍的方法中，仅适合内核版本（OS发行版本）差别不大的情况下，可以方便修改某块版本。本小节介绍如何编译非当前内核版本模块。

步骤：

1. 将待编译特定内核源码开发包拷贝到当前系统中某个目录下

如我们打算在SLES11.1 x86_64系统中编译RHEL5.5 x86_64内核模块，应先将RHEL5.5 x86_64内核开发包拷贝到SLES11.1系统中。

```
[root@localhost kernels]# pwd
/usr/src/kernels
[root@localhost kernels]# ls
2.6.18-194.el5-x86_64
[root@localhost kernels]# tar jcf 2.6.18-194.el5-x86_64.tar.bz2 2.6.18-194.el5-x86_64/
[root@localhost kernels]# ls
2.6.18-194.el5-x86_64  2.6.18-194.el5-x86_64.tar.bz2
[root@localhost kernels]#
```

```
linux:~/RHEL5.5 # ls
2.6.18-194.el5-x86_64
linux:~/RHEL5.5 # uname -r
2.6.32.12-0.7-default
linux:~/RHEL5.5 # pwd
/root/RHEL5.5
linux:~/RHEL5.5 #
```

2. 修改**Makefile**，将**KDIR**指向待编译内核开发包目录

在Makefile中，设置KDIR变量为指定内核源码目录位置。

```
ifneq    ($(KERNELRELEASE),)
obj-m    := time.o

else
KDIR    := /root/2.6.18-194.el5-x86_64/
PWD    := $(shell pwd)
default:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
    rm -r .tmp_versions *.mod.c .*.cmd *.o
endif
```

**3**、执行**make**，编译模块

执行后，生成内核模块。可以使用modinfo命令来查看新生成模块的版本号。如

#modinfo /root/programming/time.ko

编译非当前内核版本模块后，加载再次提示“Invalid module format”时，通过dmesg命令查看加载失败原因。

```
[root@localhost ~]# insmod time.ko
insmod: error inserting 'time.ko': -1 Invalid module format
[root@localhost ~]#dmesg|tail
time: version magic '2.6.18-194.el5 SMP mod_unload gcc-4.3' should be '2.6.18-194.el5 SMP
mod_unload gcc-4.1'
```

如上面的提示，我们直接include/linux/vermagic.h文件即可，将gcc版本信息值改为固定值gcc-4.1即可。

2.2内核符号表及使用

加载模块时，insmod使用公共内核符号表解析模块中未定义的符号。公共符号表中包含了所有的全局内核项（即函数和变量）的地址，内核符号表的内容全部在文件/proc/kallsyms中，可以通过cat等命令查看。内核和模块将函数、变量导出后，就成为内核符号表的一部分。

在我们编写的内核模块中，可以使用内核或其他模块定义的函数和变量，如本章示例的获取时间模块中，就调用了内核函数do_gettimeofday（）。

内核中有两个宏用来导出函数和变量：

*EXPORT_SYMBOL(symbolname)*

将函数或变量导出到所有模块

*EXPORT_SYMBOL_GPL(symbolname)*

将函数或变量仅导出到GPL模块

我们也可以在自己的模块中导出部分函数或变量，这样其他模块就可以访问这部分函数、变量。C语言用户态程序编程中，我们常会使用在其他C文件或lib库中定位的函数和变量，内核符号表和这有相似之处。

系统中所有内核和模块导出的变量和函数，就成了内核符号表，在/proc/kallsyms文

件中。

```
linux:~ # cat /proc/kallsyms |tail
ffffffffa0000310 r __kstrtab_hwmon_device_unregister  [hwmon]
ffffffffa0000300 r __kcrctab_hwmon_device_unregister  [hwmon]
ffffffffa0000340 r __func__.19301  [hwmon]
ffffffffa00000a0 t hwmon_device_register  [hwmon]
ffffffffa0000440 d __this_module  [hwmon]
ffffffffa0000168 t cleanup_module  [hwmon]
ffffffffa0000000 t hwmon_device_unregister  [hwmon]
linux:~ #
```

内核符号表中，第一列为函数或变量的在内核中的地址，第二列为符号的类型，第三列为符号名，第四列为符号所属的模块。若第四列为空，则表示该符号属于内核代码。

1. 内核符号属性

| 符号属性 | 含义 |
|---|---|
| b | 符号在未初始化数据区（BSS） |
| c | 普通符号，是未初始化区域 |
| d | 符号在初始化数据区 |
| g | 符号针对小object，在初始化数据区 |
| i | 非直接引用其他符号的符号 |
| n | 调试符号 |
| r | 符号在只读数据区 |
| s | 符号针对小object，在未初始化数据区 |
| t | 符号在代码段 |
| u | 符号未定义 |

若符号在内核中是全局性的，则属性为大写字母，如T、U等。其他符号属性含义，请参考命令nm的帮助信息。

```
00273: /* Only label it "global" if it is exported. */
00274: static void upcase_if_global(struct kallsym_iter *iter)
00275: {
00276: if (is_exported(iter->name, iter->owner))
00277: iter->type += 'A' - 'a';
00278: }
00279:
```

若打算使用内核中的符号，在模块中增加函数或变量说明即可。如：

```
00091: extern struct timespec xtime;
```

## 3模块版本控制

Linux内核版本在不变升级，内核提供的API或符号可能也随之变化。这对内核模块开发来说，是一个比较麻烦的问题，通常要适应不同的内核版本，或者只针对具体某些内核版本开发。

内核为了确保模块的函数接口与内核借口一致，采用了模块版本控制。版本控制最简单的办法就是为了内核和模块都设置一个常量，该常量会随着接口变化而不断增加。加载模块时，内核会检查模块提供的常量是否和内核版本常量相等，若不相等则拒绝加载。

采用常量的办法进行版本控制，方法简单，但不够灵活。如内核部分接口变化后，版本常量就会增加。但若某模块使用的这些接口并没有变化，也会导致驱动无法加载。基于这个原因，最恰当的方法是将单个内核API的变化考虑进去。实际的模块和内核实现无关，模块和内核关系密切的是API接口。

### 3.1checksum方法

CRC checksum原理是使用函数的参数来计算校验码，若校验码不相等，加载模块失败。

我们来看一下内核执行模块加载的函数load_module（）（文件kernel/module.c）。1767行会调用check_modstruct_version（）函数来检查struct_module符号的CRC校验码。若校验码不相等，则提示“disagrees about version of symbol struct_module”。如

*hwinc_kernel_driver: disagrees about version of symbol struct_module*

*Found checksum B6AF205C vs module F3D5F8AF*

```
01600: static struct module *load_module(void __user *umod,
01601: unsigned long len,
01602: const char __user *uargs)
01603: {
01604: Elf_Ehdr *hdr;
01605: Elf_Shdr *sechdrs;
... ...
```

```
01766: /* Check module struct version now, before we try to use module. */
01767: if (!check_modstruct_version(sechdrs, versindex, mod)) {
01768: err = -ENOEXEC;
01769: goto ↓free_hdr;
01770: }
01771:
01772: modmagic = get_modinfo(sechdrs, infoindex, "vermagic");
01773: /* This is allowed: modprobe --force will invalidate it. */
01774: if (!modmagic) {
01775: add_taint_module(mod,TAINT_FORCED_MODULE);
01776: printk(KERN_WARNING "%s: no version magic, tainting
                                        kernel.\n",
01777: mod->name);
01778: } else if (!same_magic(modmagic, vermagic )) {
01779: printk(KERN_ERR "%s: version magic '%s' should be '%s'\n",
01780: mod->name, modmagic, vermagic );
01781: err = -ENOEXEC;
01782: goto ↓free_hdr;
01783: }
```

在编译内核模块时，会生成*.mod.c文件，该文件中包含了模块中各个符号的校验码。校验码的生成，由scripts/genksyms/genksyms计算生成。

注意：scripts/genksyms/genksyms文件是在内核源码目录或内核开发包目录中，如/usr/src/linux-2.6.32.12-0.7-obj/x86_64/default/scripts/genksyms/genksyms

```
[root@localhost modules]# ls
ctime.c  ctime.ko  ctime.mod.c  ctime.mod.o  ctime.o  Makefile  Module.symvers
 [root@localhost modules]# cat ctime.mod.c
#include <linux/module.h>
#include <linux/vermagic.h>
#include <linux/compiler.h>

MODULE_INFO(vermagic, VERMAGIC_STRING);

#undef unix
struct module __this_module
__attribute__((section(".gnu.linkonce.this_module"))) = {
 .name = __stringify(KBUILD_MODNAME),
 .init = init_module,
#ifdef CONFIG_MODULE_UNLOAD
 .exit = cleanup_module,
#endif
};

static const struct modversion_info ____versions[]
__attribute_used__
__attribute__((section("__versions"))) = {
        { 0xc48fa26, "struct_module" },
        { 0xda02d67, "jiffies" },
        { 0x72270e35, "do_gettimeofday" },
        { 0xdd132261, "printk" },
};

static const char __module_depends[]
__attribute_used__
__attribute__((section(".modinfo"))) =
"depends=";
```

### 3.2vermagic

查看内核版本模块信息时，会看到vermagic一项。模块在装载时，load_module（）函数会比较（如前面代码的1772行）当前运行内核的vermagic和当前要加载的模块的vermagic比较，如果不同，则禁止加载模块。

```
mod[root@localhost ~]# modinfo fat
filename:       /lib/modules/2.6.18-128.el5/kernel/fs/fat/fat.ko
license:        GPL
srcversion:     3C5EEEB9A691E08FDAD23EC
depends:
vermagic:       2.6.18-128.el5 SMP mod_unload gcc-4.1
module_sig:
883f35049492f705cdc734e64d24fa1129fbf09f624add271d3782b2978b61a684a6d3ad98
4d7ca09f647c86194bf4a9cadc107a76d5f9896673a9
[root@localhost ~]#
```

Vermagic的的确定请参考章节2.1.1。

### 3.3内核模块版本控制使能与关闭

我们常遇到内核提示“disagrees about version of symbol struct_module”，而导致模块无法加载的情况。

#### 3.3.1内核中关闭/使能

若选择关闭内核的模块版本控制功能，则会避免出现这种情况。模块版本控制选项在内核源码配置文件.config中，注释掉CONFIG_MODVERSIONS就取消了模块版本控制。

CONFIG_MODVERSIONS=y

重新编译内核，重启即可。

### 3.3.2模块中关闭/使能版本控制

如下面的实例。虽然模块的vermagic和内核一致，但struct_module的版本号不一致。我们可以不修改当前内核，重新编译模块即可解决问题。

若去掉模块版本控制后，加载驱动导致系统死机。建议解决办法：使用待运行内核的.config配置文件覆盖模块编译指向的内核开发包（源码）.config文件。

.config配置文件的获取：（1）可以拷贝/proc/config.gz，然后解压缩，拷贝为.config；（2）若/proc/config.gz不存在，可以使用/boot/目录下对应的内核配置文件；（3）或向内核提供者获取.config配置文件。

```
[root@localhost ~]# modinfo /lib/modules/2.6.9_5-10-0-0/kernel/fs/fat/fat.ko
filename:       /lib/modules/2.6.9_5-10-0-0/kernel/fs/fat/fat.ko
license:        GPL
depends:
vermagic:       2.6.9_5-10-0-0 SMP gcc-3.4
[root@localhost ~]# cd modules/
[root@localhost modules]# modinfo ctime.ko
filename:       ctime.ko
license:        GPL
depends:
vermagic:       2.6.9_5-10-0-0 SMP gcc-3.4
[root@localhost modules]#
[root@localhost modules]# insmod ctime.ko
insmod: error inserting 'ctime.ko': -1 Invalid module format
[root@localhost modules]# dmesg
ctime: disagrees about version of symbol struct_module
[root@localhost modules]# uname -a
Linux localhost.localdomain 2.6.9_5-10-0-0 #1 SMP Fri Nov 5 17:23:43 CST 2010
x86_64 x86_64 x86_64 GNU/Linux
```

此时我们可以修改内核开发包中的模块版本控制选项，修改文件.config（在内核源码或开发包根目录下），注释掉或删除CONFIG_MODVERSIONS选项，重新编译模块即可去除模块的版本控制。

CONFIG_MODULES=y

CONFIG_OBSOLETE_MODPARM=y

#CONFIG_MODVERSIONS=y

CONFIG_MODULE_SIG=y

## 4内核模块参数

在用户执行系统命令或其他程序时，可以使用参数。内核模块也可以使用参数。

参数必须使用宏module_param（）声明，该宏定义在include/linux/moduleparam.h文件中。module_param（）需要三个参数：参数名称、类型、sysfs文件系统入口项的访问权限掩码。

模块参数的定义必须放在任何函数之外。如本章获取系统时间的模块示例，我们增加province和population两个参数（参数仅作示范，和系统时间无任何关系）。

```
00001: #include <linux/ module.h>
00002: #include <linux/ time.h>
00003: #include <linux/ moduleparam.h>
00004:
00005: static char modname[] = "time";
00006:
00007: static char *province = "Guangdong";
00008: module_param(province, charp, 0);
00009: static int population = 10000;
00010: module_param(population, int, 0);
00011:
00012: int init_module( void )
00013: {
00014: struct timeval tv;
00015: printk( "Installing %s module.", modname );
00016: do_gettimeofday(&tv);
00017: printk("\njiffies:%lu, tv.tv_sec:%lu, tv.tv_nsec:%lu ",
00018: jiffies, tv.tv_sec, tv.tv_usec);
00019:
00020: printk("\nProvince:%s, Population:%d \n", province , population );
00021:
00022: return 0;
00023: }
00024:
00025:
00026: void cleanup_module( void )
00027: {
00028: printk( "\nRemoving %s module.", modname );
00029: }
```

```
00030:
00031: MODULE_LICENSE("GPL");
00032:
```

加载模块time后，内核打印信息：

```
[root@localhost modules]# insmod time.ko province=henan population=50000
[root@localhost modules]# dmesg |tail
Installing time module.
jiffies:4312987745, tv.tv_sec:1289836355, tv.tv_nsec:118699
Province:henan, Population:50000
```

内核模块支持的参数类型如下：

bool

invbool

charp：字符串指针。内核会为用户提供的字符串自动分配内存。

int

long

short

uint：unsigned int

ulong：unsigned long

ushort：unsigned short

## 5模块入口/出口函数及其他

每个内核模块都要有初始化（入口）函数和清除（出口）函数，清除函数负责在模块被移除前注销接口并向系统返回所有资源。

在本章time模块的示例中，并没有像用户态C程序一样有main（）入口函数。time模块入口函数为init_module（），而出口函数为cleanup_module（）。

在复杂的模块中，我们可以指定模块的入口/出口函数名称。通过module_init（）和module_exit（）函数分别指定。如LSISAS1068E驱动mptsas中的入口/出口函数：

```
04828: module_init(mptsas_init);
04829: module_exit(mptsas_exit);
```

在模块中，我们还可以添加作者信息、模块描述、模块版本等信息。

MODULE_AUTHOR（）：模块作者信息

MODULE_DESCRIPTION（）：模块描述

MODULE_LICENSE（）：模块协议

MODULE_VERSION（）：模块版本

如：

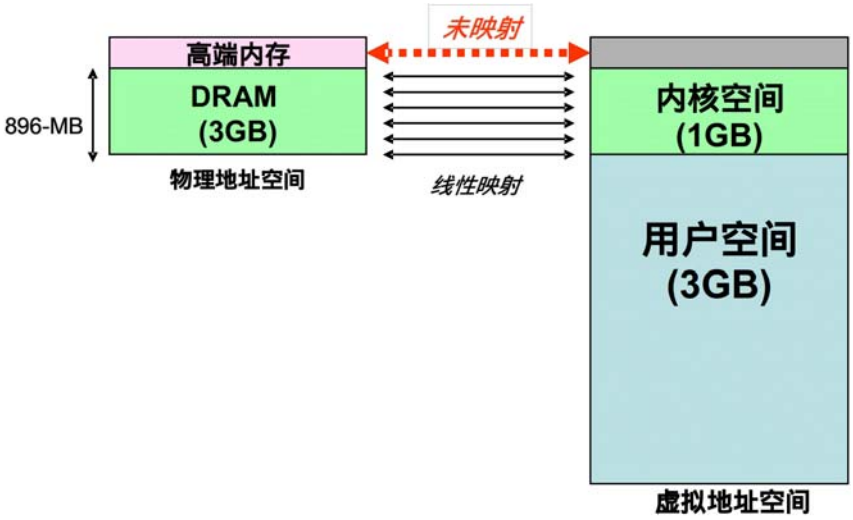
```
00070: #define my_NAME "Fusion MPT SCSI Host driver"
00071: #define my_VERSION MPT_LINUX_VERSION_COMMON
00072: #define MYNAM "mptscsih"
00073:
00074: MODULE_AUTHOR(MODULEAUTHOR);
00075: MODULE_DESCRIPTION(my_NAME);
00076: MODULE_LICENSE("GPL");
00077: MODULE_VERSION(my_VERSION);
```

## 6内核模块与用户程序区别

### 6.1用户空间与内核空间

内核空间具有最高权限，可以访问所有CPU寄存器和其他所有资源。

- 内核空间可以访问所有的CPU指令和所有的内存空间、I/O空间。
- 用户空间只能访问有限的资源，若需要特殊权限，可以通过系统调用获取相应的资源。
- 用户空间允许页面中断，而内核空间则不允许。
- 用户空间是0-3G的地址范围，内核空间是3G-4G的地址范围。
- 内核空间和用户空间是针对线性地址空间的。
- 所有内核进（线）程共用一个地址空间，而用户进程都有各自的地址空间。




1. Linux 32位系统用户空间与内核空间

### 6.2内核模块与应用程序的对比

- 内核模块具有独立的地址空间

模块运行在内核空间中。应用程序运行在用户空间中。系统软件受到保护，不允许用户程序访问。内核空间和用户空间有各自独立的内存地址空间。

- 内核模块具有更高的执行特权

运行在内核空间中的代码要比运行在用户空间中的代码具有更大的特权。

- 内核模块不按顺序执行

用户程序通常按顺序执行并且从头到尾地执行单独的任务。内核模块并不按顺序执行，它注册自己是为了服务将来的请求。

- 内核模块可以被中断

在同一时刻，可能有许多进程同时向驱动程序发出请求。中断程序可以在驱动程序正在响应系统调用时，向驱动程序发出请求。在对称多处理器（SMP）系统中，驱动程序可能在多个 CPU 上并发地执行。

- 内核模块必须是可抢占的

- 内核模块能够共享数据

一个应用程序的不同线程常常不会共享数据。与之相对应的是，组成驱动程序的数据结构和例程被所有使用驱动程序的线程所共享。驱动程序必须能够处理由多个请求导致的竞争问题。

- 错误处理

应用程序的错误导致Segmentation Fault，而内核模块的错误影响整个系统，甚至使内核

## 7常见问题处理

**1**、头文件引用

在用户程序和内核模块时，可能都会使用头文件 #include <linux/time.h>，但两者文件所在的位置是不同的。

用户态用户程序使用的time.h，在gcc库文件中，一般位置是/usr/include/linux/time.h或/usr/include/sys/time.h。

内核模块使用的time.h，在内核源码头文件中，一般位置是<内核版本>/include/linux/time.h，如/usr/src/kernels/2.6.18-128.el5-x86_64/include/linux/time.h。

**2**、提示内核**build**目录不存在

在编写好内核模块后，执行make时，有的系统会提示类似“make: *** /lib/modules/2.6.18-128.el5xen/build: No such file or directory. Stop.”错误信息。原因在于内核源码开发包没有安装。

解决办法：安装当前内核版本的源码开发包。

```
[root@localhost modules]# make
make -C /lib/modules/2.6.18-128.el5xen/build SUBDIRS=/root/modules modules
make: *** /lib/modules/2.6.18-128.el5xen/build: No such file or directory.  Stop.
make: *** [default] Error 2
[root@localhost ~]# rpm -ivh kernel-xen-devel-2.6.18-128.el5.x86_64.rpm
Preparing...                ########################################### [100%]
   1:kernel-xen-devel       ########################################### [100%]
```

**3**、模块加载提示“**Invalid Module Format**”

解决步骤：

1. 执行dmesg命令，查看模块提示Invalid Module Format的详细原因

2. 根据提示信息，结合本章提到内核模块版本号与修改一些，修复相应的错误。

**4**、模块加载提示“**Symbol not found**”

解决步骤：

1. 执行dmesg命令，查看模块哪些符号在当前系统中不存在。
2. 执行modinfo命令，查看当前模块依赖关系，并检查依赖的模块是否已加载到系统中。

**4**、模块加载提示“disagrees about version of symbol struct_module”

请参考“模块版本控制”一节。

**5**、是否有办法将模块加载到非当前内核版本中，而不重新编译模块？

在内核版本相近和CPU架构相同的情况下，如2.6.18-92.e15 i686和2.6.18-194.e15 i686内核，可以直接二进制编辑模块，修改模块的版本信息，这样就可以加载到非当前内核版本中了。