## Linux内核头文件分析

<sys/stat.h>    文件状态

<sys/times.h>    进程时间

<sys/types.h>    基本系统数据类型

<sys/wait.h>    进程控制

<sys/resource.h> 资源操作

asm/目录下主要用于存放与计算机体系结构密切相关的函数声明或数据结构的头文件。例如Intel CPU 端口IO汇编宏文件io.h、中断描述符设置汇编宏头文件system.h等。linux/目录下是Linux内核程序使用的一些头文件。其中包括调度程序使用的头文件sched.h、内存管理头文件mm.h和终端管理数据结构文件tty.h等。而sys/目录下存放着几个与内核资源相关头文件。不过从0.98版开始，内核目录树下sys/目录中的头文件被全部移到了linux/目录下。

## 1、EXPORT\_SYMBOL的作用是什么？

EXPORT\_SYMBOL标签内定义的函数或者符号对全部内核代码公开，不用修改内核代码就可以在您的内核模块中直接调用，即使用EXPORT\_SYMBOL可以将一个函数以符号的方式导出给其他模块使用。  
这里要和System.map做一下对比：  
System.map 中的是连接时的函数地址。连接完成以后，在2.6内核运行过程中，是不知道哪个符号在哪个地址的。  
EXPORT\_SYMBOL的符号，是把这些符号和对应的地址保存起来，在内核运行的过程中，可以找到这些符号对应的地址。而模块在加载过程中，其本质就是能动态连接到内核，如果在模块中引用了内核或其它模块的符号，就要EXPORT\_SYMBOL这些符号，这样才能找到对应的地址连接。

2、使用方法  
   第一、在模块函数定义之后使用EXPORT\_SYMBOL（函数名）  
   第二、在掉用该函数的模块中使用extern对之声明  
   第三、首先加载定义该函数的模块，再加载调用该函数的模块

另外，在编译调用某导出函数的模块时，往往会有WARNING: "＊＊＊＊" [＊＊＊＊＊＊＊＊＊＊] undefined!  
使用dmesg命令后会看到相同的信息。开始我以为只要有这个错误就不能加载模块，后来上网查了一下，发现这主要是因为在编译连接的时候还没有和内核打交道，当然找不到symbol了，但是由于你生成的是一个内核模块，所以LD不提示error，而是给出一个warning，寄希望于在insmod的时候，内核能够把这个symbol连接上。

## 被调函数原型分析

#### （1）情况一：宏调用批量生成内核函数

#define build\_mmio\_read(name, size, type, reg, barrier) \

static inline type name(const volatile void \_\_iomem \*addr) \

{ type ret; asm volatile("mov" size " %1,%0":reg (ret) \

:"m" (\*(volatile type \_\_force \*)addr) barrier); return ret; }

build\_mmio\_read(readb, "b", unsigned char, "=q", :"memory")

build\_mmio\_read(readw, "w", unsigned short, "=r", :"memory")

build\_mmio\_read(readl, "l", unsigned int, "=r", :"memory")

生成函数readb，readw，readl

build\_mmio\_read(\_\_readb, "b", unsigned char, "=q", )

build\_mmio\_read(\_\_readw, "w", unsigned short, "=r", )

build\_mmio\_read(\_\_readl, "l", unsigned int, "=r", )

生成函数\_\_readb，\_\_readw，\_\_readl

在.int文件中解析的函数声明function\_decl中定位的位置，函数声明开始位置和结束位置是一样的，因此这样的函数比较时直接提取这一样对比即可。

#### （2）情况二：inline内核函数

**con\_free\_unimap []**

**con\_set\_default\_unimap []**

原型头文件-》include/linux/vt\_kern.h

static inline int [con\_set\_default\_unimap](http://elixir.free-electrons.com/linux/v3.5.4/ident/con_set_default_unimap)(struct [vc\_data](http://elixir.free-electrons.com/linux/v3.5.4/ident/vc_data) \*[vc](http://elixir.free-electrons.com/linux/v3.5.4/ident/vc))

{

return 0;

}

static inline void [con\_free\_unimap](http://elixir.free-electrons.com/linux/v3.5.4/ident/con_free_unimap)(struct [vc\_data](http://elixir.free-electrons.com/linux/v3.5.4/ident/vc_data) \*[vc](http://elixir.free-electrons.com/linux/v3.5.4/ident/vc))

{

}

内联函数有些类似于宏。内联函数的代码会被直接嵌入在它被调用的地方2，调用几次就嵌入几次，没有使用call指令。这样省去了函数调用时的一些额外开销，比如保存和恢复函数返回地址等，可以加快速度。不过调用次数多的话，会使可执行文件变大，这样会降低速度。相比起宏来说，内核开发者一般更喜欢使用内联函数。因为内联函数没有长度限制，格式限制。编译器还可以检查函数调用方式，以防止其被误用。

static inline的内联函数，一般情况下不会产生函数本身的代码，而是全部被嵌入在被调用的地方。如果不加static，则表示该函数有可能会被其他编译单元所调用，所以一定会产生函数本身的代码。所以加了static，一般可令可执行文件变小。内核里一般见不到只用inline的情况，而都是使用static inline。

#### （3）情况三：只有声明语句的内核函数

**request\_resource []**

原型头文件-》include/linux/ioport.h

extern int [request\_resource](http://elixir.free-electrons.com/linux/latest/ident/request_resource)(struct [resource](http://elixir.free-electrons.com/linux/latest/ident/resource) \*root, struct [resource](http://elixir.free-electrons.com/linux/latest/ident/resource) \*[new](http://elixir.free-electrons.com/linux/latest/ident/new));

（只有声明，定义在[kernel/resource.c, line 254 (as a function)](http://elixir.free-electrons.com/linux/v3.5.4/source/kernel/resource.c#L254)， EXPORT\_SYMBOL()导出此符号）

Extern修饰的函数编译时会忽略，链接时才进行符号替换，和包含头文件的做法一样，我们include 了<stdio.h>，这里声明了好多函数，就是告诉编译器，操作系统中这些函数都已经实现了，而extern也是说，内核已经拥有此代码的二进制实现

**vc\_resize []**

原型头文件-》include/linux/vt\_kern.h

int [vc\_resize](http://elixir.free-electrons.com/linux/v3.5.4/ident/vc_resize)(struct [vc\_data](http://elixir.free-electrons.com/linux/v3.5.4/ident/vc_data) \*[vc](http://elixir.free-electrons.com/linux/v3.5.4/ident/vc), unsigned int [cols](http://elixir.free-electrons.com/linux/v3.5.4/ident/cols), unsigned int [lines](http://elixir.free-electrons.com/linux/v3.5.4/ident/lines));

(只有声明，定义在[drivers/tty/vt/vt.c, line 978 (as a function)](http://elixir.free-electrons.com/linux/v3.5.4/source/drivers/tty/vt/vt.c#L978)，EXPORT\_SYMBOL()导出符号)

## Catgs

获取函数定义列表：

ctags -xu --c-kinds=+px   file.c

## 局限性Limitation

[1] for #ifdef ,different computer have different choices when compile source generated AST

## 自定义函数被它本身调用次数分析

vgacon\_text\_force

text\_mode

no\_scroll

write\_vga 3

vga\_set\_mem\_top 4

vgacon\_restore\_screen 2

vgacon\_scrolldelta 1

vgacon\_startup

vgacon\_init

vgacon\_deinit

vgacon\_build\_attr

vgacon\_invert\_region

vgacon\_set\_cursor\_size 9

vgacon\_cursor

vgacon\_doresize 2

vgacon\_switch

vga\_set\_palette 2

vgacon\_set\_palette

vga\_vesa\_blank 1

vga\_vesa\_unblank 1

vga\_pal\_blank 1

vgacon\_blank

vgacon\_do\_font\_op 2

vgacon\_adjust\_height 1

vgacon\_font\_set

vgacon\_font\_get

vgacon\_resize

vgacon\_set\_origin 1

vgacon\_save\_screen

vgacon\_scroll

vgacon\_dummy