modinfo相关的C特性

--------#、##和attribute

**一、背景介绍**

在阅读linux内核代码过程中，很多宏跟踪到底层都会用到#、##和attribute这些符号，为了方便代码的阅读，特对这些内容专门做次介绍。

#和##属于C语言的预处理运算符。需要和#define一起使用。

attribute是GNU C扩展的关键字，只有在gcc中才能使用，是linux内核中非常重要和常用的一个关键字。

**二、知识点：**

1.“#”:

“#”运算符，通常称为stringize运算符，使得在它后面的变元被预处理为带引号的字符串。例如

/\*\*\*\*\*\*test.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define MSG(s) # s

int main()

{

printf(“%s\n”,MSG(string));

}

使用gcc –E test.c查看预处理之后的结果，如下：

int main()

{

printf(“%s\n”,”string”);

}

2.“##”

“##”运算符，称为pasting，用于连接两个符号。例如：

/\*\*\*\*\*test.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CONCAT(x,y) x ## y

int main()

{

char filename[]=”test.c”;

printf(“%s\n”,CONCAT(file , name));

return 0;

}

使用gcc –E test.c查看预处理之后的结果，如下：

int main()

{

char filename[]=”test.c”;

printf(“%s\n”,filename);

return 0;

}

3.attribute关键字

attribute关键字用来修改函数、变量和类型的属性。用法：\_\_attribute\_\_((属性))，可修改的属性很多，详情可查看《gcc the Complete Reference》，实例：

假设有下面代码：

/\*\*\*\*\*\*test1.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

int a;

int b;

return 0;

}

我们知道，因为变量a和b没有被使用，所以在编译过程中，编译器会提示警告。若将代码如下修改：

/\*\*\*\*\*\*test1.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

int a \_\_attribute\_\_((unused));

int b;

return 0;

}

再编译的时候，变量a就不会出现未被使用的警告了。unused属性修饰变量，表示忽略掉“未被使用”的警告。

再看一个例子：

/\*\*\*\*\*\*test2.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int a;

int b;

int main()

{

return 0;

}

编译之后生成test2.o文件，使用objdump –t test2.o可看到，变量a和b都分配在COM段中：



修改变量a的section属性，代码如下：

/\*\*\*\*\*\*test2.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int a;

int b \_\_attribute\_\_((section(“.modinfo”)));

int main()

{

return 0;

}

再次编译之后生成test2.o文件，使用objdump –t test2.o可看到，变量a依然在COM段中，而变量b被分配到了我们指定的.modinfo段：



**三、示例演示**

**示例1：**

有了上面的知识基础，我们下来看下linux内核中如何应用这些内容的。

1. 在<linux/module.h>中，有下面这些宏被定义，可以在代码中直接使用。

MODULE\_ALIAS(alias)

MODULE\_LICENSE(license);

MODULE\_AUTHOR(description)

MODULE\_DESCRIPTION(parm\_desc)

MODULE\_VERSION(version)

MODULE\_FIRMWARE(firmware)

2. 继续跟踪这些宏，发现这些宏都基于MODULE\_INFO宏构造，而MODULE\_INFO宏又由\_\_MODULE\_INFO而来，如下：

#define MODULE\_ALIAS(\_alias) MODULE\_INFO(alias, \_alias)

#define MODULE\_LICENSE(\_license) MODULE\_INFO(license, \_license)

#define MODULE\_AUTHOR(\_author) MODULE\_INFO(author, \_author)

#define MODULE\_DESCRIPTION(\_description) MODULE\_INFO(description, \_description )

#define MODULE\_VERSION(\_version) MODULE\_INFO(version, \_version)

#define MODULE\_FIRMWARE(\_firmware) MODULE\_INFO(firmware, \_firmware)

#define MODULE\_INFO(tag, info) \_\_MODULE\_INFO(tag, tag, info)

3. 在<linux/moduleparam.h>中,可以看到\_\_MODULE\_INFO的实现代码，如下：

#ifdef MODULE

#define \_\_\_module\_cat(a,b) \_\_mod\_ ## a ## b

#define \_\_module\_cat(a,b) \_\_\_module\_cat(a,b)

#define \_\_MODULE\_INFO(tag, name, info) \

static const char \_\_module\_cat(name,\_\_LINE\_\_)[] \

\_\_attribute\_used\_\_ \

\_\_attribute\_\_((section(".modinfo"),unused)) = \_\_stringify(tag) "=" info

#else /\* !MODULE \*/

#define \_\_MODULE\_INFO(tag, name, info)

#endif

4. 综上：MODULE\_LICENSE(“Dual BSD/GPL”);

=> MODULE\_INFO(licese , “Dual BSD/GPL”);

=> \_\_MODULE\_INFO(license, license, “Dual BSD/GPL”);

=> static const char license\_\_LINE\_\_[] \_\_attribute\_\_((section(“.modinfo”),unused)) = ”license” “=” “Dual BSD/GPL”;

//上句表示定义了一个静态的，只读的字符数组license\_\_LINE\_\_，并且赋初始字符串”license=Dual BSD/GPL”。

忽略掉属性，假设此句代码在100行，则上句变为。

static const char license100[] = “license=Dual BSD/GPL”; //这句比较熟悉吧。

属性\_\_attribute\_\_((section(“.modinfo”),unused))修饰变量license100，表示变量编译后放在内存.modinfo段，并且编译阶段忽略掉unused警告。

**示例2：**

我们也可以自己直接将代码定义到.modinfo段，然后用modinfo也可以查看出。代码如下：

/\*\*\*\*\*\*\*\*\*\*\*test.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

const char myCountry[] \_\_attribute\_\_((section(“.modinfo”)))=”myCountry=中华人民共和国”;

int main()

{

return 0;:

}

gcc test.c //编译为可执行程序

用modinfo就可以查看上面代码编译出的可执行程序了。

modinfo a.out

