tanglinux 只专注于arm和linux系统平台的研究与开发

个人资料

tanglinux

访问： 584355次
积分： 4478
等级： BLOG 5
排名： 第5809名

原创： 69篇 转载： 0篇
译文： 0篇 评论： 101条

Linux内核中的常用宏container_of其实很简单

标签： linux内核 struct compiler structure float ubuntu

2011-11-27 19:50 19213人阅读 评论(6) 收藏 举报

分类： linux内核修炼之其它（12）

开发平台：Ubuntu11.04

编 译器：gcc version 4.5.2 (Ubuntu/Linaro4.5.2-8ubuntu4)

Container_of在Linux内核中是一个常用的宏，用于从包含在某个结构中的指针获得结构本身的指针，通俗地讲就是通过结构体变量中某个成员的首地址进而获得整个结构体变量的首地址。

Container_of的定义如下：

```cpp
#define container_of(ptr, type, member) ({      \
    const typeof( ((type *)0)->member ) *__mptr = (ptr);    \
    (type *)( (char *)__mptr - offsetof(type,member) );})
```

其实它的语法很简单，只是一些指针的灵活应用，它分两步：

第一步，首先定义一个临时的数据类型（通过typeof(((type *)0)->member)获得）与ptr相同的指针变量__mptr，然后用它来保存ptr的值。

第二步，用(char *)__mptr减去member在结构体中的偏移量，得到的值就是整个结构体变量的首地址（整个宏的返回值就是这个首地址）。

其中的语法难点就是如何得出成员相对结构体的偏移量?

通过例子说明，如清单1：

```cpp
/* linux-2.6.38.8/include/linux/compiler-gcc4.h */
#define __compiler_offsetof(a,b) __builtin_offsetof(a,b)

/* linux-2.6.38.8/include/linux/stddef.h */
#undef offsetof
#ifdef __compiler_offsetof
#define offsetof(TYPE,MEMBER) __compiler_offsetof(TYPE,MEMBER)
#else
#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *)0)->MEMBER)
#endif

#include <stdio.h>

struct test_struct {
    int num;
```

(51170)
Linux进程管理之task_str
(33295)
Ubuntu操作系统“Failed t
(23264)
详解Linux内核红黑树算
(21559)
Linux内核中的常用宏con
(19207)
拯救无法启动的虚拟机文
(16530)
散列表的基本概念及其运
(16074)
Linux进程管理之task_str
(10270)
使用CodeViz生成C/C++
(10190)

评论排行

详解二叉查找树算法的实 (19)
常用库之四：zlib的交叉 (10)
常用库之六：ibfontconfig (8)
p++和++p的区别 (8)
详解Linux内核红黑树算 (8)
常用库之五：libtiff的交叉 (7)
Linux内核中的常用宏con (6)
使用CodeViz生成C/C++ (4)
常用库之二：libfreetype (3)
制作文件系统之四：Ubu (3)

最新评论

详解Linux内核红黑树算法的实现
sinat_35351858: 感谢博主的分享！想问下rb_set_parent函数的用法static inline void rb...

常用库之一：libjpeg的交叉编译
waruqi: 可以用xmake 快速快速跨平台构建 http://xmake.io

散列表的基本概念及其运算
JQ_AK47: 感谢分享

散列表的基本概念及其运算
十二期刘超: 很有帮助，感谢分享、

详解二叉查找树算法的实现
K_天道酬勤: 楼主写得很棒，这篇博客用java实现二叉搜索树和平衡二叉树，里面还有练习题 如果读者有兴趣可以看看：...

Linux进程管理之task_struct结构
likelei123: 赞一个

详解二叉查找树算法的实现
kongyue08: 就是说s -> lchild = s -> rchild = NULL; 左右孩子lchild和rc...

详解二叉查找树算法的实现
tanglinux: @GG_and_DD:函数的英文名为function，它也可以翻译成“功能”，所以最好一个函数实现一...

详解二叉查找树算法的实现
给我丶鼓励: 插入那里用了两个函数。我觉得有点复杂了。完全可以写一个函数。

详解二叉查找树算法的实现
rqzrqh: @bochuan007:设计模式中的 vistor模式

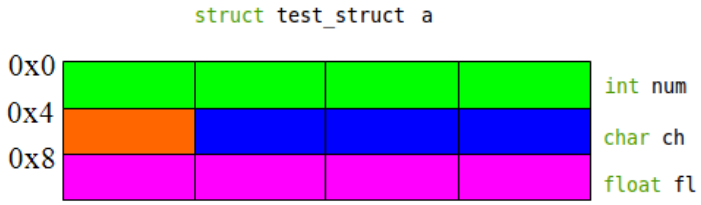

```
16.     char ch;
17.     float fl;
18. };
19.
20. int main(void)
21. {
22.     printf("offsetof(struct test_struct, num) = %d\n",
23.             offsetof(struct test_struct, num));
24.
25.     printf("offsetof(struct test_struct,  ch) = %d\n",
26.             offsetof(struct test_struct, ch));
27.
28.     printf("offsetof(struct test_struct,  fl) = %d\n",
29.             offsetof(struct test_struct, fl));
30.
31.     return 0;
32. }
```

说明，__builtin_offsetof(a,b)是GCC的内置函数，可认为它的实现与((size_t) &((TYPE *)0)->MEMBER)这段代码是一致的。

例子输出结果：

```
[cpp]
01. offsetof(struct test_struct, num) = 0
02. offsetof(struct test_struct,  ch) = 4
03. offsetof(struct test_struct,  fl) = 8
```

其中代码难以理解的地方就是它灵活地运用了0地址。如果觉得&((struct test_struct *)0)->ch这样的代码不好理解，那么我们可以假设在0地址分配了一个结构体变量struct test_struct a，然后定义结构体指针变量p并指向a（struct test_struct *p = &a），如此我们就可以通过&p->ch获得成员ch的地址。由于a的首地址为0x0，所以成员ch的首地址为0x4。



最后通过强制类型转换（size_t）把一个地址值转换为一个整数。

分析完container_of的定义，接下来举两个例子来体会一下它的使用方法。

正确的例子，如清单2：

```
[cpp]
01. /* linux-2.6.38.8/include/linux/compiler-gcc4.h */
02. #define __compiler_offsetof(a,b) __builtin_offsetof(a,b)
03.
04. /* linux-2.6.38.8/include/linux/stddef.h */
05. #undef offsetof
06. #ifdef __compiler_offsetof
07. #define offsetof(TYPE,MEMBER) __compiler_offsetof(TYPE,MEMBER)
08. #else
09. #define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *)0)->MEMBER)
10. #endif
11.
12. /* linux-2.6.38.8/include/linux/kernel.h *
13.  * container_of - cast a member of a structure out to the containing structure
14.  * @ptr: the pointer to the member.
15.  * @type:   the type of the container struct this is embedded in.
16.  * @member:    the name of the member within the struct.
17.  *
18.  */
19. #define container_of(ptr, type, member) ({      \
20.     const typeof( ((type *)0)->member ) *__mptr = (ptr);    \
21.     (type *)( (char *)__mptr - offsetof(type,member) );})
22.
```

```c
#include <stdio.h>

struct test_struct {
    int num;
    char ch;
    float fl;
};

int main(void)
{
    struct test_struct init_test_struct = { 99, 'C', 59.12 };

    char *char_ptr = &init_test_struct.ch;

    struct test_struct *test_struct = container_of(char_ptr, struct test_struct, ch);

    printf(" test_struct->num = %d\n test_struct->ch = %c\n test_struct->fl = %f\n",
        test_struct->num, test_struct->ch, test_struct->fl);

    return 0;
}
```

例子输出结果：

```
[cpp]

test_struct->num = 99
test_struct->ch = C
test_struct->fl = 59.119999
```

不适当的例子，如清单3：

```
[cpp]

/* linux-2.6.38.8/include/linux/compiler-gcc4.h */
#define __compiler_offsetof(a,b) __builtin_offsetof(a,b)

/* linux-2.6.38.8/include/linux/stddef.h */
#undef offsetof
#ifdef __compiler_offsetof
#define offsetof(TYPE,MEMBER) __compiler_offsetof(TYPE,MEMBER)
#else
#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *)0)->MEMBER)
#endif

/* linux-2.6.38.8/include/linux/kernel.h *
 * container_of - cast a member of a structure out to the containing structure
 * @ptr: the pointer to the member.
 * @type:   the type of the container struct this is embedded in.
 * @member:    the name of the member within the struct.
 *
 */
#define container_of(ptr, type, member) ({      \
    const typeof( ((type *)0)->member ) *__mptr = (ptr);    \
    (type *)( (char *)__mptr - offsetof(type,member) );})

#include <stdio.h>

struct test_struct {
    int num;
    char ch;
    float fl;
};

int main(void)
{
    char real_ch = 'A';
    char *char_ptr = &real_ch;

    struct test_struct *test_struct = container_of(char_ptr, struct test_struct, ch);

    printf(" char_ptr = %p  test_struct = %p\n\n", char_ptr, test_struct);

    printf(" test_struct->num = %d\n test_struct->ch = %c\n test_struct->fl = %f\n",
        test_struct->num, test_struct->ch, test_struct->fl);

    return 0;
```

```
44. }
```

例子输出结果：

```
[cpp]
01. char_ptr = 0xbfb72d7f  test_struct = 0xbfb72d7b
02.
03. test_struct->num = -1511000897
04. test_struct->ch = A
05. test_struct->fl = 0.000000
```

注意，由于这里并没有一个具体的结构体变量，所以成员num和fl的值是不确定的。

上一篇 例解GNU C之表达式中的复合语句
下一篇 制作文件系统之一：安装交叉编译工具链

我的同类文章

linux内核修炼之其它（12）

- 详解Linux内核红黑树算法的... 2012-04-11 阅读 21556
- Linux内核中的PID散列表实例 2012-03-27 阅读 5409
- 详解Linux内核双向循环链表... 2012-02-27 阅读 3404
- 通过proc文件系统输出必要的.. 2012-02-05 阅读 1866
- 详解likely和unlikely函数 2012-01-05 阅读 9361
- Linux内核的通知链机制 2012-03-31 阅读 4839
- 详解Linux内核双向循环链表... 2012-02-27 阅读 4212
- 通过proc文件系统输出必要的.. 2012-02-05 阅读 1569
- 通过proc文件系统输出必要的.. 2012-02-05 阅读 2936
- 内核移植之编译初体验 2011-11-20 阅读 1378

更多文章

猜你在找

查看评论

5楼 neo_peng 2015-02-09 18:58发表

请问为什么不能写成这样的形式呢?
#define container_of(ptr, type, member) ((type *)((char *)(ptr) - offsetof(type, member)))

4楼 ghost_like_8 2013-03-06 08:36发表

這裡請注意：只是說NULL指針不能被解引用（*NULL）而不是說NULL指針不能不使用。懂?

3楼 a617505352 2012-11-13 11:15发表

(type *)((char *)__mptr - offsetof(type,member));
不明白为什么要把__mptr转换成char *再相减？能否赐教1 2?

Re: tanglinux 2012-11-14 09:33发表

这是指针变量的运算问题，举个简单的例子：
int a;
char *p = (char *)&a;
int *q = &a;
这时候，p-1和q-1的值是不是相等呢?
肯定不是，如果&a的值为0xbffe67b4，那么p-1为0xbffe67b3，而q-1则为0xbffe67b0。
因此，指针变量加减n实际上是加减n个数据类型（p为char，q为int）的长度。
回复a617505352：

2楼 tanglinux 2012-06-05 13:33发表

没错，0地址当然不能被访问，这句话的意思是定义了一个跟member数据类型一样的指针变量__mptr，也就是说如果member的数据类型为int，那么typeof(((type *) NULL)->member) *__mptr = (ptr); 就等价于int *__mptr = (ptr);，整个typeof(((type *) NULL)->member) 语句就是为了获得成员member的数据类型。

1楼 liuchen180126 2012-06-04 15:42发表

不错的文章，但是有一点不明白，就是关于0地址的使用，0地址不是不能被访问吗?
typeof(((type *)0)->member) *__mptr = (ptr);

岂不是相当于

typeof(((type *) NULL)->member) *__mptr = (ptr);

能否赐教1，2?

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC
coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 | 江苏乐知网络技术有限公司