Tekkaman Ninja

tekkamanninja.blog.chinaunix.net

tekkamanninj

博客访问： 75916
博文数量： 263
博客积分： 15936
博客等级： 上将
技术积分： 13951
用 户 组： 普通用户
注册时间： 2007-03-27 11:22

Linux设备驱动程序学习（7）-内核的数据类型

2007-11-09 16:12:28

分类： LINUX

Linux设备驱动程序学习（7）-内核的数据类型

由于前面的学习中有用到 第十一章 内核数据结构类型 的知识，所以我先看了。要点如下：

将linux 移植到新的体系结构时，开发者遇到的若干问题都与不正确的数据类型有关。坚持使用严格的数据类型和使用 -Wall -Wstrict-prototypes 进行编译可能避免大部分的 bug。

内核数据使用的数据类型主要分为 3 个类型：标准 C 语言类型、确定大小的类型和特定内核对象的类型。

标准 C 语言类型

当需要“一个2字节填充符”或“用一个4字节字串来代表某个东西”，就不能使用标准C语言类型，因为在不同的体系结构，C 语言的数据类型所占的空间大小不同。后面的datasize 程序实验展示了用户空间各种 C 的数据类型在当前平台所占空间的大小。而且有的构架，内核空间和用户空间的C 数据类型所占空间大小也可能不同。kdatasize模块显示了当前模块的内核空间C 数据类型所占空间大小。

尽管概念上地址是指针，但使用一个无符号整型可以更好地实现内存管理；内核把物理内存看成一个巨型数组，内存地址就是该数组的索引。 我们可以方便地对指针取值，但直接处理内存地址时，我们几乎从不会以这种方式对他取值。使用一个整数类型避免了这种取值，因此避免了 bug。所以，利用至少在 Linux 目前支持的所有平台上，指针和长整型始终是相同大小的这一事实，内核中内存地址常常是 unsigned long。

C99 标准定义了 intptr_t 和 uintptr_t 类型，它们是能够保存指针值的整型变量。但没在 2.6 内核中几乎没使用。

确定大小的类型

当需要知道你定义的数据的大小时，可以使用内核提供的下列数据类型（所有的数据声明在 <asm/types.h>，被包含在 <linux/types.h> ）：

```
u8; /* unsigned byte (8 bits) */
u16; /* unsigned word (16 bits) */
u32; /* unsigned 32-bit value */
u64; /* unsigned 64-bit value */
/*虽然很少需要有符号类型，但是如果需要，只要用 s 代替 u*/
```

若一个用户空间程序需要使用这些类型，可在符号前加一个双下划线：__u8和其它类型是独立于 __KERNEL__ 定义的。

这些类型是 Linux 特定的，它们妨碍了移植软件到其他的 Unix 机器。新的编译器系统支持 C99-标准类型，如 uint8_t 和 uint32_t。若考虑移植性，使用这些类型比 Linux特定的变体要好。

接口特定的类型（_t 类型）

内核中最常用的数据类型由它们自己的 typedef 声明，阻止了任何移植性问题。“接口特定（interface-specific）”由某个库定义的一种数据类型，以便为了某个特定的数据结构提供接口。很多 _t 类型在 <linux/types.h> 中定义。

注意：近来已经很少定义新的接口特定的类型。有许多内核开发者已经不再喜欢使用 typedef 语句，他们宁愿看到代码中直接使用的真实类型信息。很多老的接口特定类型在内核中保留，他们不会很快消失。

即使没有定义接口特定类型，也应该始终是用和内核其他部分保持一致、适当的数据类型。只要驱动使用了这种“定制”类型的函数，但又不遵照约定，编译器会发出警告，这时使用 -Wall 编译器选项并小心

去除所有的警告，就可以确信代码的可移植性了。

_t 类型的主要问题是：打印它们时，常常不容易选择正确的 printk 或 printf 格式。打印接口特定的数据的最好方法是：将其强制转换为可能的最大类型(常常是 long 或 unsigned long) 并用相应的格式打印。

其他移植性问题

移植的一个通常规则是：避免使用显式的常量值，要使用预处理宏使常量值参数化。

时间间隔

当处理时间间隔时，不要假定每秒的jiffies个数，不是每个 Linux 平台都以固定的速度运行.当计算时间间隔时，要使用 HZ （ 每秒的定时器中断数 ）来标定你的时间。s3c2410的HZ值默认为200。

页大小

当使用内存时，记住一个内存页是 PAGE_SIZE 字节，不是 4KB。相关的宏定义是 PAGE_SIZE 和 PAGE_SHIT（包含将一个地址移位来获得它的页号的位数），在 <asm/page.h> 中定义。如果用户空间程序需要这些信息，可以使用 getpagesize 库函数。

若一个驱动需要 16 KB 来暂存数据，一个可移植得解决方法是 get_order:

```
#include <asm/page.h>
int order = get_order(16*1024);
buf = get_free_pages(GFP_KERNEL, order);/*get_order 的参数必须是 2 的幂*/
```

字节序

不要假设字节序。 代码应该编写成不依赖所操作数据的字节序的方式。

头文件 <asm/byteorder.h> 定义：

```
#ifdef __ARMEB__
#include <linux/byteorder/big_endian.h>
#else
#include <linux/byteorder/little_endian.h>
#endif
```

在<linux/byteorder/big_endian.h>中定义了__BIG_ENDIAN ，而在<linux/byteorder/little_endian.h>中定义了__LITTLE_ENDIAN，这些依赖处理器的字节序当处理字节序问题时,需要编码一堆类似 #ifdef __LITTTLE_ENDIAN 的条件语句。

但是还有一个更好的方法：Linux 内核有一套宏定义来处理处理器字节序和特定字节序之间的转换。例如:

```
u32 cpu_to_le32 (u32);
u32 le32_to_cpu (u32);
/*这些宏定义将一个CPU使用的值转换成一个无符号的32位小头数值，无论 CPU 是大端还是小端，
也不管是不是32 位处理器。在没有转换工作需要做时，返回未修改的值。*/

/*有很多类似的函数在 <linux/byteorder/big_endian.h> 和
<linux/byteorder/little_endian.h> 中定义*/
```

数据对齐

编写可移植代码而值得考虑的最后一个问题是如何访问未对齐的数据。存取不对齐的数据应当使用下列宏:

```
#include <asm/unaligned.h>
get_unaligned(ptr);
put_unaligned(val, ptr);
```

这些宏是无类型的，并对各总数据项，不管是 1、2、4或 8 个字节，他们都有效，并且在所有内核版本中都有定义。

关于对齐的另一个问题是数据结构的跨平台移植性。同样的数据结构在不同的平台上可能被不同地编译。为了编写可以跨体系移植的数据结构，应当始终强制数据项的自然对齐。自然对齐（natural alignment）指的是：数据项大小的整数倍的地址上存储数据项。 应当使用填充符避免强制自然对齐时编译器移动数据结构的字段，在数据结构中留下空洞。

dataalign 程序实验展示了编译器如何强制对齐。

为了目标处理器的良好性能，编译器可能悄悄地插入填充符到结构中，来保证每个成员是对齐的。若定义一个和设备要求的结构体相匹配结构，自动填充符会破坏这个意图。解决这个问题的方法是告诉编译器这个结构必须是"紧凑的"，不能增加填充符。例如下列的定义：

```
struct
{
        u16 id;
        u64 lun;
        u16 reserved1;
        u32 reserved2;
}
__attribute__ ((packed)) scsi;

/*如果在 64-位平台上编译这个结构，若没有 __attribute__ ((packed))，lun 成员可能在前面被添加 2 个或 6 个填充符字节。指针和错误值*/
```

你还可以在利用ARM9和USB摄像头进行视频采集的servfox源代码的spcaframe.h头文件中找到这种方法的实际应用：

```
struct frame_t{
    char header[5];
    int nbframe;
    double seqtimes;
    int deltatimes;
    int w;
    int h;
    int size;
    int format;
    unsigned short bright;
    unsigned short contrast;
    unsigned short colors;
    unsigned short exposure;
    unsigned char wakeup;
    int acknowledge;
    } __attribute__ ((packed));
struct client_t{
    char message[4];
    unsigned char x;
    unsigned char y;
    unsigned char fps;
    unsigned char updobright;
    unsigned char updocontrast;
    unsigned char updocolors;
    unsigned char updoexposure;
    unsigned char updosize;
    unsigned char sleepon;
    } __attribute__ ((packed));
```

指针和错误值

许多内核接口通过将错误值编码到指针值中来返回错误信息。这样的信息必须小心使用，因为它们的返回值不能简单地与 NULL 比较。为帮助创建和使用这类接口，<linux/err.h>提供了这样的函数：

```
void *ERR_PTR(long error);/*将错误值编码到指针值中，error 是常见的负值错误码*/
long IS_ERR(const void *ptr); /*测试返回的指针是不是一个错误码*/
long PTR_ERR(const void *ptr); /*抽取实际的错误码，只有在IS_ERR 返回一个真值时使用，否则一个有效指针*/
```

链表

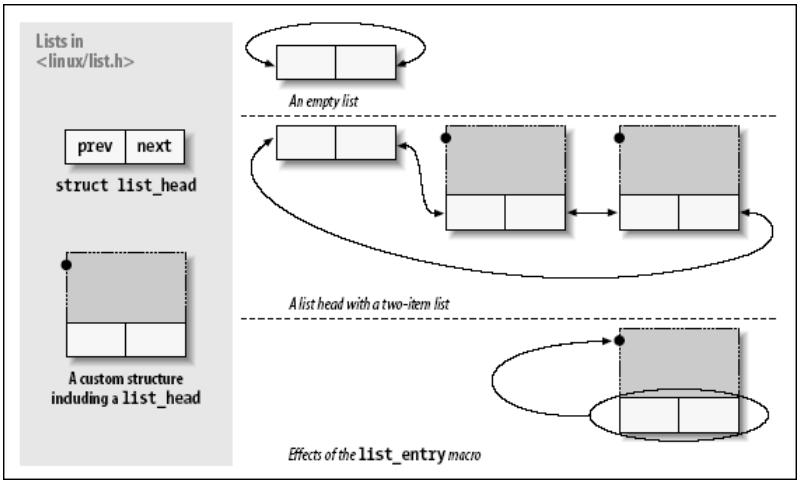
操作系统内核常需要维护数据结构的链表。Linux 内核已经同时有几个链表实现。为减少复制代码的数量，内核已经创建了一个标准环形双向链表，并鼓励需要操作链表的人使用这个设施.

使用链表接口时，应当记住列表函数没做加锁。若驱动可能同一个列表并发操作，就必须实现一个锁方案。

为使用链表机制，驱动必须包含文件 <linux/list.h>，它定义了一个简单的list_head 类型 结构:

```
struct list_head { struct list_head *next, *prev; };
```

实际代码中使用的链表几乎总是由某个结构类型组成, 每个结构描述链表中的一项. 为使用 Linux 链表，只需嵌入一个 list_head 在构成在这个链表的结构里面。链表头常常是一个独立的 list_head 结构。下图显示了这个简单的 struct list_head 是如何用来维护一个数据结构的列表的.




```
/*链表头必须在使用前初始化,有两种形式： */
/*一是运行时初始化:*/
struct list_head todo_list;
INIT_LIST_HEAD(&todo_list);
/*二是编译时初始化:*/
LIST_HEAD(todo_list);

list_add(struct list_head *new, struct list_head *head);
/*在紧接着链表 head 后面增加新项 。注意：head 不需要是链表名义上的头; 如果你传递一个 list_head 结构, 它在链表某处的中间, 新的项紧靠在它后面。 因为 Linux 链表是环形的, 链表头通常和任何其他的项没有区别*/
list_add_tail(struct list_head *new, struct list_head *head);
/*在给定链表头前面增加新项，即在链表的尾部增加一个新项。*/

list_del(struct list_head *entry);
list_del_init(struct list_head *entry);
/*给定的项从队列中去除。 如果入口项可能注册在另外的链表中, 你应当使用 list_del_init, 它重新初始化这个链表指针.*/

list_move(struct list_head *entry, struct list_head *head);
list_move_tail(struct list_head *entry, struct list_head *head);
/*给定的入口项从它当前的链表里去除并且增加到 head 的开始。为安放入口项在新链表的末尾, 使用 list_move_tail 代替*/

list_empty(struct list_head *head);
/*如果给定链表是空, 返回一个非零值.*/

list_splice(struct list_head *list, struct list_head *head);
/*将 list 紧接在 head 之后来连接 2 个链表.*/

/*list_entry是将一个 list_head 结构指针转换到一个指向包含它的结构体的指针 。看了源码你就会发现，似曾相识。是的，其实在模块的open方法中已经用到了container_of。list_entry的变体好有很多，看源码就知道了*/
```

```
#define list_entry(ptr, type, member) \
 container_of(ptr, type, member)
```

ARM9实验板实验

datasize实验

实验中有用到uname函数，介绍如下（载自《UNIX环境高级编程》，第6章系统数据文件和信息 6.8 系统标识）：

> POSIX.1定义了uname函数，它返回与主机和操作系统有关的信息。
>
> #include <sys/utsname.h>
>
> int uname(struct utsname *n a m e) ;
>
> 返回：若成功则为非负值，若出错则为-1
>
> 通过该函数的参数向其传递一个utsname结构的地址，然后该函数填写此结构。POSIX.1只定义了该结构中至少需提供的字段(它们都是字符数组)，而每个数组的长度则由实现确定。某些实现在该结构中提供了另外一些字段。在历史上，系统 V为每个数组分配9个字节，其中有1个字节用于字符串结束符(null字符)。
>
> struct utsname {
>
> char sysname[9]; /* name of the operating system */
>
> char nodename[9]; /* name of this node */
>
> char release[9]; /* current release of operating system */
>
> char version[9]; /* current version of this release */
>
> char machine[9]; /* name of hardware type */
>
> } ;
>
> utsname结构中的信息通常可用uname(1)命令打印。

实验程序源码链接：**datasize**

kdatasize模块实验

实验中有用到utsname函数，源码如下：

```
//<linux/utsname.h>
struct new_utsname {
    char sysname[65];
    char nodename[65];
    char release[65];
    char version[65];
    char machine[65];
    char domainname[65];
};

static inline struct new_utsname *utsname(void)
{
    return &current->nsproxy->uts_ns->name;
}
```

实验模块源码链接：kdatasize

kdataalign模块实验

具体试验原理请看源码

实验模块源码链接：kdataalign

实验现象：

```
[Tekkaman2440@SBC2440V4]#cd /tmp/
[Tekkaman2440@SBC2440V4]#./datasize
arch Size: char short int long ptr long-long u8 u16 u32 u64
armv4tl 1 2 4 4 4 8 1 2 4 8
[Tekkaman2440@SBC2440V4]#cd /lib/modules/
[Tekkaman2440@SBC2440V4]#insmod kdatasize.ko
arch Size: char short int long ptr long-long u8 u16 u32 u64
armv4tl 1 2 4 4 4 8 1 2 4 8
```

```
insmod: cannot insert 'kdatasize.ko': No such device (-1): No such device
[Tekkaman2440@SBC2440V4]#insmod kdataalign.ko
arch Align: char short int long ptr long-long u8 u16 u32 u64
armv4tl 1 2 4 4 4 4 1 2 4 4
insmod: cannot insert 'kdataalign.ko': No such device (-1): No such device
```


上一篇：Linux设备驱动程序学习（6）-高级字符驱动程序操作 [（3）设备文件的访问控制]
下一篇：各种 IDE 线缆比较

0

相关热门文章

如何看待操作系统的用户空间和...	linux 常见服务端口	移植 ushare 到开发板
linux内核空间和用户空间详解...	【ROOTFS搭建】busybox的httpd...	系统提供的库函数存在内存泄漏...
Linux用户空间与内核空间...	xmanager 2.0 for linux配置	linux虚拟机 求教
强类型语言、弱类型语言、静态...	什么是shell	初学UNIX环境高级编程的，关于...
Linux内核container_of详解（...	linux socket的bug??	chinaunix博客什么时候可以设...

给主人留下些什么吧！~~

评论热议

请登录后评论。
登录 注册