

Tekkaman Ninja

tekkamanninja.blog.chinaunix.net

Linux我的梦想，我的未来！ 本博客的原创文章的内容会不定期更新或修正错误！转载文章都会注明出处，若有侵权，请即时同我联系，我一定马上删除！！原创文章版权所有！如需转载，请注明出处： tekkamanninja.blog.chinaunix.net ，谢谢合作！！ 拒绝一切广告性质的评论，一经发现立即举报并删除！

首页 | 博文目录 | 关于我



tekkamanninj

博客访问： 75904
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

加关注 短消息
论坛 加好友

个人简介

Fedora-ARM

文章分类

- 全部博文（263）
- Red Hat（2）
 - 代码管理（6）
 - 感悟（3）
 - Linux调试技术（2）
 - MaxWit（1）
 - Linux设备驱动程（41）
 - Android（20）
 - neo freerunner（2）
 - 计算机硬件技术（9）
 - 网络（WLAN or LA（8）
 - 励志（7）
 - ARM汇编语言（1）
 - Linux操作系统的（15）
 - Linux内核研究（38）
 - ARM-Linux应用程（19）
 - 建立根文件系统（4）
 - Linux内核移植（14）
 - Bootloader（45）
 - 建立ARM-Linux交（7）
 - 未分配的博文（19）

文章存档

2014年（1）

Linux设备驱动程序学习（0） - Hello, world模块 2007-10-25 10:23:07

分类： LINUX

Linux设备驱动程序学习（0） —设备驱动介绍& Hello, world! 模块

设备驱动程序的作用

设备驱动程序就是这个进入Linux内核世界的大门。设备驱动程序在Linux内核中扮演着特殊的角色。它是一个独立的“黑盒子”，使某个特定硬件响应一个定义好的内部编程接口，这些接口完全隐藏了设备的工作细节。用户的操作通过一组标准化的调用执行，而这些调用独立于特定的驱动程序。将这些调用映射到作用于实际硬件的设备特有操作上，则是设备驱动程序的任务。

设备驱动的分类

字符设备：字符(char)设备是个能够像字节流（类似文件）一样被访问的设备。字符设备驱动程序通常至少要实现 open、close、read和write系统调用。

块设备：一个块设备驱动程序主要通过传输固定大小的数据来访问设备。块设备和字符设备的区别仅仅在于内核内部管理数据的方式，也就是内核及驱动程序之间的软件接口，而这些不同对用户程序是透明的。在内核中，和字符驱动程序相比，块驱动程序具有完全不同的接口。

网络接口：任何网络事务都经过一个网络接口形成，即一个能够和其他主机交换数据的设备。它可以是个硬件设备，但也可能是个纯软件设备。访问网络接口的方法仍然是给它们分配一个唯一的名字（比如eth0），但这个名字在文件系统中不存在对应的节点。内核和网络设备驱动程序间的通信，完全不同于内核和字符以及块驱动程序之间的通信，内核调用一套和数据包传输相关的函数而不是read、write等。

驱动模块的特点

（1）驱动模块运行在内核空间，运行时不能依赖于任何标准C库等应用层的库、模块，所以在写驱动时所调用的函数只能是作为内核一部分的函数，即使用“EXPORT_SYMBOL”导出的函数。

→insmod使用公共内核符号表来解析模块中未定义的符号。公共内核符号表中包含了所有的全局内核项（即函数和变量的地址），这是实现模块化驱动程序所必须的。

→Linux使用模块层叠技术，我们可以将模块划分为多个层，通过简化每个层可缩短开发周期。如果一个模块需要向其他模块导出符号，则使用下面的宏：

```
EXPORT_SYMBOL(name);  
EXPORT_SYMBOL_GPL(name);
```

符号必须在模块文件的全局变量部分导出，因为这两个宏将被扩展为一个特殊变量的声明，而该变量必须是全局的。

（2）驱动模块和应用程序的一个重要不同是：应用程序退出时可不管资源释放或者其他的清除工作，但模块的退出函数必须仔细撤销初始化函数所作的一切，否则，在系统重新引导之前某些东西就会残留在系统中。

（3）处理器的多种工作模式（级别）其实就是为了操作系统的用户空间和内核空间设计的。在Unix类的操作系统中只用到了两个级别：最高和最低级别。

（4）要十分注意驱动程序的并发处理。

（5）内核API中具有双下划线（_ _）的函数，通常是接口的底层组件，应慎用。


- 2013年（3）
- 2012年（61）
- 2011年（66）
- 2010年（27）
- 2009年（30）
- 2008年（23）
- 2007年（52）


我的朋友



小蜗牛快



cfm5538



jikaishi



shizhenc


pxy05215


李怀远


yan19900


wkm81018


xiousi

最近访客


apang199


appcount


zaichu


lhxzui


小蜗牛快


小尾巴鱼


erain_30


hushup


wilfred_

订阅

推荐博文

- linux 3.x的 通用时钟架构 ...
- SCN的相关解析
- Flash驱动学习
- 浅谈nagios之state type和 no...
- DB2（Linux 64位）安装教程...
- insert语句造成latch:library...
- 2014.06.13 网络公开课《让我...
- MySQL Slave异常关机的处理（...
- 巧用shell脚本分析数据库用户...
- 查询linux, HP-UX的cpu信息...

热词专题

- linux系统权限修复——学生误...
- Modbus协议使用
- linux
- busybox原理
- php环境搭建教程

（6）内核代码不能实现浮点数运算。参考资料

料：<http://blog.chinaunix.net/u/30180/showart.php?id=1421920>

模块结构介绍

利用Linux设备驱动程序的第一个例程：Hello World模块了解内核驱动模块的结构。

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

static int hello_init(void)
{
    printk(KERN_ALERT "Hello, Tekkaman Ninja ! \n");
    return 0;
}

static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, Tekkaman Ninja ! \n Love Linux !Love ARM ! Love KeKe !\n");
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("Dual BSD/GPL");
```

1. 所有模块代码中都包含一下两个头文件：

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

2. 所有模块代码都应该指定所使用的许可证：

```
MODULE_LICENSE("Dual BSD/GPL");
```

此外还有可选的其他描述性定义：

```
MODULE_AUTHOR("");
MODULE_DESCRIPTION("");
MODULE_VERSION("");
MODULE_ALIAS("");
MODULE_DEVICE_TABLE("");
```

上述MODULE_声明习惯上放在文件最后。

3. 初始化和关闭

初始化的实际定义通常如下：

```
static int __init initialization_function(void)
{
    /*初始化代码*/
}

module_init(initialization_function)
```

清除函数的实际定义通常如下：

```
static int __exit cleanup_function(void)
{
    /*清除代码*/
}

module_exit(cleanup_function)
```

4. 一个简单的Makefile文件：

```
KERNELDIR = /home/tekkaman/working/SBC2440/linux-2.6.22.2
PWD := $(shell pwd)
INSTALLDIR = /home/tekkaman/working/rootfs/lib/modules
CROSS_COMPILE = arm-9tdmi-linux-gnu-
CC = $(CROSS_COMPILE)gcc
obj-m := hello.o
.PHONY: modules modules_install clean
```

```
modules:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

modules_install:
    cp hello.ko $(INSTALLDIR)

clean:
    rm -rf *.o *~ core .depend *.cmd *.ko *.mod.c .tmp_versions
```

`obj-m := hello.o`

代表了我们要构造的模块名为`hell.ko`，`make` 会在该目录下自动找到`hell.c`文件进行编译。如果 `hello.o` 是由其他的源文件生成（比如`file1.c`和`file2.c`）的，则在下面加上（注意红色字体的对应关系）：

```
hello-objs := file1.o file2.o .....
```

`$(MAKE) -C $(KERNELDIR) M=$(PWD) modules`

`-C $(KERNELDIR)` 指定了内核源代码的位置，其中保存有内核的顶层`makefile`文件。

`M=$(PWD)` 指定了模块源代码的位置

`modules`目标指向`obj-m`变量中设定的模块。

5. 编译模块

`make modules`、`make modules_install`。

```
[root@Tekkaman-Ninja Helloworld]# make modules
make -C /home/tekkaman/working/SBC2440/linux-2.6.22.2 M=/home/tekkaman/working/Linuxdriver/Helloworld modules
make[1]: Entering directory `/home/tekkaman/working/SBC2440/linux-2.6.22.2'
  CC [M] /home/tekkaman/working/Linuxdriver/Helloworld/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/tekkaman/working/Linuxdriver/Helloworld/hello.mod.o
  LD [M] /home/tekkaman/working/Linuxdriver/Helloworld/hello.ko
make[1]: Leaving directory `/home/tekkaman/working/SBC2440/linux-2.6.22.2'
[root@Tekkaman-Ninja Helloworld]# make modules_install
cp hello.ko /home/tekkaman/working/rootfs/lib/modules
[root@Tekkaman-Ninja Helloworld]#
```

6. 在开发板上的操作：

```
[Tekkaman2440@SBC2440V4]# cd /lib/modules/
[Tekkaman2440@SBC2440V4]# ls
cs89x0.ko hello.ko p80211.ko prism2_usb.ko
[Tekkaman2440@SBC2440V4]# insmod hello.ko
Hello, Tekkaman Ninja !
[Tekkaman2440@SBC2440V4]# lsmod
Module Size Used by Not tainted
hello 1376 0
[Tekkaman2440@SBC2440V4]# rmmod hello
Goodbye, Tekkaman Ninja !
Love Linux ! Love ARM ! Love KeKe !
[Tekkaman2440@SBC2440V4]# lsmod
Module Size Used by Not tainted
[Tekkaman2440@SBC2440V4]#
```

Linux内核模块的初始化出错处理一般使用“`goto`”语句。

通常情况下很少使用“`goto`”，但在出错处理是（可能是唯一的情况），它却非常有用。在 大二学习C语言时，老师就建议不要使用“`goto`”，并说很少会用到。在这里也是我碰到的第一个建议使用“`goto`”的地方。“在追求效率的代码中使用`goto`语句仍是最好的错误恢复机制。”——《Linux设备驱动程序（第3版）》以下是初始化出错处理的推荐代码示例：

```
struct something *item1;
```

```
struct somethingelse *item2;
int stuff_ok;

void my_cleanup(void)
{
    if (item1)
        release_thing(item1);
    if (item2)
        release_thing2(item2);
    if (stuff_ok)
        unregister_stuff();
    return;
}

int __init my_init(void)
{
    int err = -ENOMEM;
    item1 = allocate_thing(arguments);
    item2 = allocate_thing2(arguments2);
    if (!item1 || !item2)
        goto fail;
    err = register_stuff(item1, item2);
    if (!err)
        stuff_ok = 1;
    else
        goto fail;
    return 0; /* success */
fail:
    my_cleanup();
    return err;
}
```

模块参数

内核允许对驱动程序指定参数，而这些参数可在装载驱动程序模块时改变。

以下是我的实验程序：

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h>

static char *whom = "Tekkaman";
static int howmany = 1;
static int param_array[] = {0,0,0,0};
static int param_array_nr;
module_param(howmany, int, S_IRUGO);
module_param(whom, charp, S_IRUGO);
module_param_array(param_array, int, &param_array_nr, S_IRUGO);

static int hello_init(void)
{
    int i;

    printk(KERN_ALERT "Hello, Linux ! \n");

    for (i = 0; i < howmany; i++) {
        printk(KERN_ALERT "(%d) Hello, %s\n", i, whom);
    }

    for (i = 0; i < param_array_nr; i++) {
        printk(KERN_ALERT "param_array[%d] : %d \n", i, param_array[i]);
    }

    return 0;
}

static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, Linux ! \n");
}

module_init(hello_init);
module_exit(hello_exit);

EXPORT_SYMBOL(hello_init);

MODULE_DESCRIPTION("hello_linux test module");
MODULE_ALIAS("hello_world");
```

```
MODULE_INFO(tekkaman, "ninja");
MODULE_VERSION("v1.0");
MODULE_AUTHOR("Tekkaman");
MODULE_LICENSE("Dual BSD/GPL");
```

实验结果是：

```
root@tekkaman:~# ls
hello_linux.ko
root@tekkaman:~# insmod hello_linux.ko param_array=9,8,7
Hello, Linux !
(0) Hello, Tekkaman
param_array[0] : 9
param_array[1] : 8
param_array[2] : 7
root@tekkaman:~# rmmod hello_linux.ko
Goodbye, Linux !
root@tekkaman:~# insmod hello_linux.ko param_array=9,8
Hello, Linux !
(0) Hello, Tekkaman
param_array[0] : 9
param_array[1] : 8
root@tekkaman:~# rmmod hello_linux.ko
Goodbye, Linux !
root@tekkaman:~# insmod hello_linux.ko param_array=9,8,7,6
Hello, Linux !
(0) Hello, Tekkaman
param_array[0] : 9
param_array[1] : 8
param_array[2] : 7
param_array[3] : 6
root@tekkaman:~# rmmod hello_linux.ko
Goodbye, Linux !
root@tekkaman:~# insmod hello_linux.ko param_array=9,8,7,6,5
param_array: can only take 4 arguments
hello_linux: `9' invalid for parameter `param_array'
insmod: error inserting 'hello_linux.ko': -1 Invalid parameters
```

module_param_array(param_array , int , ¶m_array_nr , S_IRUGO);” 中
¶m_array_nr是用于保存模块加载时输入数组参数的成员数目的。
以前我错误的以为这个是输入参数，其实这个用于内核模块加载器输出的。

- （15）“#include <linux/sched.h>” 最重要的头文件之一。包含驱动程序使用的大部分内核API的定义，包括睡眠函数以及各种变量声明。
- （16）“#include <linux/version.h>” 包含所构造内核版本信息的头文件。

在学习过程中找到了几篇很好的参考文档：

- （1）第一章 模块（Modules） URL: <http://greenlinux.blogcn.com/diary,103232026.shtml>
- （2）《从 2.4 到 2.6: Linux 内核可装载模块机制的改变对设备驱动的影响》
URL: <http://www.ibm.com/developerworks/cn/linux/l-module26/>
- （3）《Linux2.6内核驱动移植参考》
URL: http://blog.chinaunix.net/u1/40912/showart_377391.html

以上就是我对《Linux设备驱动程序（第3版）》的《第二章 构造和运行模块》的学习总结。

阅读(17423) | 评论(0) | 转发(38) |

上一篇: Linux设备驱动程序学习（1）-字符设备驱动程序
下一篇: Linux设备驱动程序学习（2）-调试技术

相关热门文章

- | | |
|----------------------------|-----------------------|
| linux 常见服务端口 | 移植 ushare 到开发板 |
| 【ROOTFS搭建】busybox的httpd... | 系统提供的库函数存在内存泄漏... |
| xmanager 2.0 for linux配置 | linux虚拟机 求救 |
| 什么是shell | 初学UNIX环境高级编程的，关于... |
| 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](#)

[关于我们](#) | [关于IT168](#) | [联系方式](#) | [广告合作](#) | [法律声明](#) | [免费注册](#)

Copyright 2001-2010 ChinaUnix.net All Rights Reserved 北京皓辰网域网络信息技术有限公司. 版权所有

感谢所有关心和支持过ChinaUnix的朋友们

京ICP证041476号 京ICP证060528号