

Linux/Android开发记录

学习、记录、分享Linux/Android开发技术

目录视图

摘要视图

RSS 订阅

个人资料



liuhaoyutz



访问: 80608次
积分: 1673分
排名: 第7877名

原创: 83篇 转载: 0篇
译文: 0篇 评论: 59条

博客声明

本博客文章均为原创，欢迎转载交流。转载请注明出处，禁止用于商业目的。

博客专栏



Android应用开发学习笔记
文章: 30篇
阅读: 17067



LDD3源码分析
文章: 17篇
阅读: 29965

文章分类

LDD3源码分析 (18)

ADC驱动 (1)

触摸屏驱动 (1)

LCD驱动 (1)

Linux设备模型 (8)

USB驱动 (0)

Android架构分析 (12)

Cocos2d-x (1)

C陷阱与缺陷 (3)

Android应用开发 (30)

Linux设备驱动程序架构分析 (8)

有奖征资源，博文分享有内涵

5月推荐博文汇总

大数据读书汇--获奖名单公布

2014 CSDN博文大赛

LDD3源码分析之并发与竞态

分类: LDD3源码分析

2012-03-22 16:19

1162人阅读

评论(0)

收藏

举报

struct

module

semaphore

活动

structure

file

作者: 刘昊昱

博客: <http://blog.csdn.net/liuhaoyutz>

编译环境: Ubuntu 10.10

内核版本: 2.6.32-38-generic-pae

LDD3源码路径: examples/scull/main.c examples/misc-modules/complete.c

本文分析LDD3第五章并发与竞态相关代码，本章代码涉及两个内容，一个是信号量，另一个是completion。

一、scull中信号量的使用

在scull_write函数中有如下代码片断：

```
[cpp]
01. 364 if (!dptr->data[s_pos]) {
02. 365     dptr->data[s_pos] = kmalloc(quantum, GFP_KERNEL);
03. 366     if (!dptr->data[s_pos])
04. 367         goto out;
05. 368 }
```

假设有两个进程A和B同时在尝试向同一个scull设备的相同偏移量写入数据，而且在同一时刻达到364行的if判断。如果dptr->data[s_pos]的值为NULL，两个进程都决定分配内存，则A、B进程都会把kmalloc的返回值赋值给dptr->data[s_pos]。显然，后赋值的那个进程会覆盖先赋值的进程的所赋的值。并且造成先赋值进程所分配的内存无法再找回来。

上述情况就是一种竞态。为了避免竞态的发生，scull使用了信号量。

scull设备用scull_dev结构体表示，该结构体在scull.h中定义如下：

```
[cpp]
01. 87struct scull_dev {
02. 88     struct scull_qset *data; /* Pointer to first quantum set */
03. 89     int quantum; /* the current quantum size */
04. 90     int qset; /* the current array size */
05. 91     unsigned long size; /* amount of data stored here */
06. 92     unsigned int access_key; /* used by sculluid and scullpriv */
07. 93     struct semaphore sem; /* mutual exclusion semaphore */
08. 94     struct cdev cdev; /* Char device structure */
09. 95};
```

最新评论

- LDD3源码分析之内存映射
wzw88486969:
@fjlhlonng:unsigned long offset
= vma->vm_pgoff <v...
Linux设备驱动程序架构分析之I2
teamos:看了你的i2c的几篇文章，真是受益匪浅，虽然让自己写还是ie不出来。非常感谢
LDD3源码分析之块设备驱动程序
electfan2011: 感谢楼主的精彩讲解，受益匪浅啊！
LDD3源码分析之slab高速缓存
donghuwuwei: 省去了不少修改的时间，真是太好了
LDD3源码分析之时间与延迟操作
donghuwuwei: jit.c代码需要加上一个头文件。
LDD3源码分析之slab高速缓存
捧灰: 今天学到这里了，可是为什么我没有修改源码一遍就通过了额。。。内核版本是2.6.18-53.el5-x...
LDD3源码分析之字符设备驱动程序
捧灰: 参照楼主的博客在自学~谢谢楼主！
LDD3源码分析之调试技术
fantasyhujian: 分析的很清楚，赞一个！
LDD3源码分析之字符设备驱动程序
fantasyhujian: 有时间再好好读读，真的分析的不错！
LDD3源码分析之hello.c与Makef
fantasyhujian: 写的很详细，对初学者很有帮助！！

阅读排行

- LDD3源码分析之字符设: (3143)
LDD3源码分析之hello.c: (2701)
S3C2410驱动分析之LCI (2527)
Linux设备模型分析之kse (2435)
LDD3源码分析之内存映! (2336)
LDD3源码分析之与硬件! (2333)
Android架构分析之Andrc (2093)
LDD3源码分析之时间与; (1987)
LDD3源码分析之poll分析 (1972)
S3C2410驱动分析之AD (1948)

评论排行

- LDD3源码分析之字符设: (12)
S3C2410驱动分析之触摸 (7)
LDD3源码分析之内存映! (5)
LDD3源码分析之hello.c: (4)
Linux设备模型分析之kot (4)
LDD3源码分析之slab高; (4)
S3C2410驱动分析之LCI (3)
LDD3源码分析之阻塞型I (3)
LDD3源码分析之时间与; (3)
LDD3源码分析之poll分析 (2)

文章存档

- 2014年06月 (1)
2014年05月 (4)
2014年04月 (1)

在scull_dev结构体中，93行定义的sem成员，就是信号量，因为每个scull_dev结构体代表一个scull设备，所以每个scull设备都有一个专用的信号量。

如果要对使用的scull设备使用一个全局的信号量也是可以的，但是，不同的scull设备并不共享资源，没有理由让一个进程在其他进程访问不同的scull设备时等待。为每个scull设备提供专用的信号量，允许不同设备上的操作可以并行处理，从而提高性能。

信号量在使用之前必须先初始化，scull在模块初始化函数scull_init_module中执行下面的循环完成对所有scull设备专用信号量的初始化：

```
[cpp]
01. 648 /* Initialize each device. */
02. 649 for (i = 0; i < scull_nr_devs; i++) {
03. 650     scull_devices[i].quantum = scull_quantum;
04. 651     scull_devices[i].qset = scull_qset;
05. 652     init_MUTEX(&scull_devices[i].sem);
06. 653     scull_setup_cdev(&scull_devices[i], i);
07. 654 }
```

这个for循环每循环一次，完成对一个scull设备的初始化，其中652行，调用init_MUTEX每个设备专用的信号量(互斥体)进行初始化。要注意，信号量必须在设备被注册到系统中之前完成初始化，否则会出现竞态。scull设备的注册是在653行的scull_setup_cdev函数中完成的，所以在这个函数调用之前，我们完成了对信号量的初始化。

在使用信号量之前，首先要明确什么是需要用信号量保护的资源，然后，我们才能用信号量保证对这些资源的互斥访问。对于scull设备来说，所有的信息都保存在scull_dev结构体中，因此，scull_dev就是我们要保护的资源。

在main.c文件在有很多地方使用了信号量来保证对scull_dev的互斥访问。或者说，凡是要改变scull_dev结构体内容的地方，都必须加锁，防止竞态。

例如，在scull_write函数中，有如下语句：

```
[cpp]
01. 346 if (down_interruptible(&dev->sem))
02. 347     return -ERESTARTSYS;
```

346行，调用down_interruptible(&dev->sem)进行加锁，注意，要对down_interruptible的返回值进行检查，如果返回0，说明说明加锁成功了，可以开始操作受保护的资源scull_dev，反之，如果down_interruptible返回非0值，说明是在等待过程中被中断了，这时要退出并返回-ERESTARTSYS，交给系统处理。

给信号量加锁后，不管scull_write能否完成其工作，都必须释放信号量，代码如下：

```
[cpp]
01. 384 out:
02. 385 up(&dev->sem);
03. 386 return retval;
```

385行，释放信号量。

至此，信号量相关的代码就分析完了。

二、completion的使用

驱动开发中，有时我们需要在当前线程(A)之外创建另外一个线程(B)执行某个活动，然后线程(A)等待该活动结束，待活动结束后，线程(A)再继续向下执行。例如，这个活动可以是某种硬件操作。这种情况下，要实现新老线程的同步，可以使用completion接口。

注意，上面所说的情况，使用信号量也能实现同步，但信号量并不适合。因为在通常的使用中，如果试图锁定某个信号量，一般来说，都能加锁成功。如果存在对信号量的严重竞争，性能将受很大影响。这时，我们就需要检查一下我们的加锁操作设计是不是有问题了。信号量对“可用”情况已经做了

2014年01月 (1)

2013年12月 (6)

展开

文章搜索

推荐文章

大量优化。对于上面所说的情况，如果用信号量实现同步，则加锁的线程几乎总是要等待，造成系统性能下降。

completion是一种轻量级的机制，它允许一个线程告诉另外一个线程某个工作已经完成。

等待completion使用如下函数：

```
void wait_for_completion(struct completion *c);
```

相应的，completion事件可以通过如下函数触发：

```
void complete(struct completion *c);
```

```
void complete_all(struct completion *c);
```

如果有多个线程在等待同一个completion事件，complete函数只唤醒一个等待线程，而complete_all函数将唤醒所有等待线程。

LDD3提供了一个complete模块演示completion机制的用法。这个模块代码不多，下面列出其源码：

```
[cpp]
01. 1/*
02. 2 * complete.c -- the writers awake the readers
03. 3 *
04. 4 * Copyright (C) 2003 Alessandro Rubini and Jonathan Corbet
05. 5 * Copyright (C) 2003 O'Reilly & Associates
06. 6 *
07. 7 * The source code in this file can be freely used, adapted,
08. 8 * and redistributed in source or binary form, so long as an
09. 9 * acknowledgment appears in derived source files. The citation
10. 10 * should list that the code comes from the book "Linux Device
11. 11 * Drivers" by Alessandro Rubini and Jonathan Corbet, published
12. 12 * by O'Reilly & Associates. No warranty is attached;
13. 13 * we cannot take responsibility for errors or fitness for use.
14. 14 *
15. 15 * $Id: complete.c,v 1.2 2004/09/26 07:02:43 gregkh Exp $
16. 16 */
17. 17
18. 18#include <linux/module.h>
19. 19#include <linux/init.h>
20. 20
21. 21#include <linux/sched.h> /* current and everything */
22. 22#include <linux/kernel.h> /* printk() */
23. 23#include <linux/fs.h> /* everything... */
24. 24#include <linux/types.h> /* size_t */
25. 25#include <linux/completion.h>
26. 26
27. 27MODULE_LICENSE("Dual BSD/GPL");
28. 28
29. 29static int complete_major = 0;
30. 30
31. 31DECLARE_COMPLETION(comp);
32. 32
33. 33ssize_t complete_read (struct file *filp, char __user *buf, size_t count, loff_t *pos)
34. 34{
35. 35    printk(KERN_DEBUG "process %i (%s) going to sleep\n",
36. 36            current->pid, current->comm);
37. 37    wait_for_completion(&comp);
38. 38    printk(KERN_DEBUG "awoken %i (%s)\n", current->pid, current->comm);
39. 39    return 0; /* EOF */
40. 40}
41. 41
42. 42ssize_t complete_write (struct file *filp, const char __user *buf, size_t count,
43. 43                        loff_t *pos)
44. 44{
45. 45    printk(KERN_DEBUG "process %i (%s) awakening the readers...\n",
46. 46            current->pid, current->comm);
47. 47    complete(&comp);
48. 48    return count; /* succeed, to avoid retrial */
49. 49}
50. 50
51. 51
52. 52struct file_operations complete_fops = {
53. 53    .owner = THIS_MODULE,
54. 54    .read = complete_read,
```

```
55. 55 .write = complete_write,
56. 56};
57. 57
58. 58
59. 59int complete_init(void)
60. 60{
61. 61     int result;
62. 62
63. 63     /*
64. 64      * Register your major, and accept a dynamic number
65. 65      */
66. 66     result = register_chrdev(complete_major, "complete", &complete_fops);
67. 67     if (result < 0)
68. 68         return result;
69. 69     if (complete_major == 0)
70. 70         complete_major = result; /* dynamic */
71. 71     return 0;
72. 72}
73. 73
74. 74void complete_cleanup(void)
75. 75{
76. 76     unregister_chrdev(complete_major, "complete");
77. 77}
78. 78
79. 79module_init(complete_init);
80. 80module_exit(complete_cleanup);
```

79行，指定模块初始化函数为complete_init。

80行，指定模块清理函数是complete_cleanup。

我们先看模块初始化函数complete_init的实现：

66行，使用老的字符设备注册函数register_chrdev注册字符设备，因为在29行设置complete_major为0，所以是由系统动态分配主设备号；模块名称为“complete”；模块对应的文件操作函数集是complete_fops。

52 - 56行，定义了complete_fops，指定读写操作分别是complete_read和complete_write。

下面看complete_read的实现：

在打印即将进入睡眠的信息后，complete_read在37行调用wait_for_completion(&comp)，进入睡眠，即等待completion“comp”。“comp”是在31行用DECLARE_COMPLETION(comp)创建的。如果等待的completion发生了，complete_read函数将再次打印已被唤醒相关信息。

也就是说，任何进程读取模块设备文件，都会进入睡眠等待。

再来看complete_write的实现：

首先打印提示信息，然后在47行调用complete(&comp)触发completion事件，相应会唤醒一个在等待“comp”的进程。

可以有多个进程进行读操作，这些读进程都会进入睡眠等待，当有执行写操作的进程时，只有一个等待进程会被唤醒，但是哪个进程，不能确定。

为测试complete模块，我改写了LDD3提供的scull_load和scull_unload脚本，命名为complete_load和complete_unload。

complete_load脚本的内容如下所示：

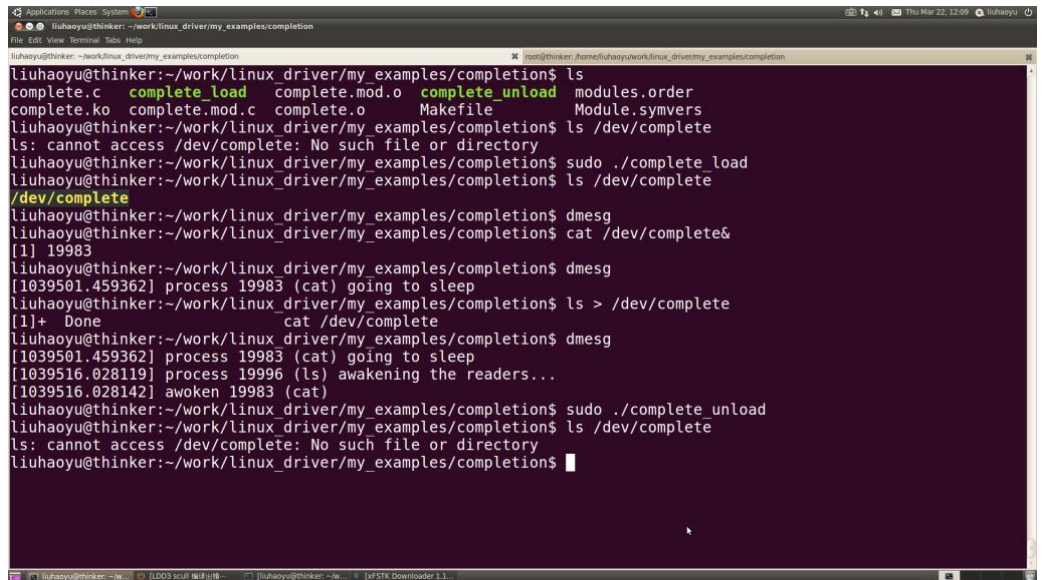
```
[cpp]
01. #!/bin/sh
02. # $Id: complete_load,v 1.4 2004/11/03 06:19:49 rubini Exp $
03. module="complete"
04. device="complete"
05. mode="666"
06.
07. # Group: since distributions do it differently, look for wheel or use staff
08. if grep -q '^staff:' /etc/group; then
09.     group="staff"
10. else
11.     group="wheel"
```

```
12. fi
13.
14. # invoke insmod with all arguments we got
15. # and use a pathname, as insmod doesn't look in . by default
16. /sbin/insmod ./module.ko $* || exit 1
17.
18. # retrieve major number
19. major=$(awk "\$2==\"$module\" {print \$1}" /proc/devices)
20.
21. # Remove stale nodes and replace them, then give gid and perms
22. # Usually the script is shorter, it's scull that has several devices in it.
23.
24. rm -f /dev/${device}
25. mknod /dev/${device} c $major 0
26.
27. chgrp $group /dev/${device}
28. chmod $mode /dev/${device}
```

complete_unload脚本的内容如下所示:

```
[cpp]
01. #!/bin/sh
02. module="complete"
03. device="complete"
04.
05. # invoke rmmod with all arguments we got
06. /sbin/rmmod $module $* || exit 1
07.
08. # Remove stale nodes
09.
10. rm -f /dev/${device}
```

complete模块的测试过程如下:



```
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$ ls
complete.c  complete_load  complete.mod.o  complete_unload  modules.order
complete.ko  complete.mod.c  complete.o      Makefile          Module.symvers
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$ ls /dev/complete
ls: cannot access /dev/complete: No such file or directory
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$ sudo ./complete_load
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$ ls /dev/complete
/dev/complete
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$ dmesg
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$ cat /dev/complete&
[1] 19983
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$ dmesg
[1039501.459362] process 19983 (cat) going to sleep
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$ ls > /dev/complete
[1]+  Done                  cat /dev/complete
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$ dmesg
[1039501.459362] process 19983 (cat) going to sleep
[1039516.028119] process 19996 (ls) awakening the readers...
[1039516.028142] awoken 19983 (cat)
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$ sudo ./complete_unload
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$ ls /dev/complete
ls: cannot access /dev/complete: No such file or directory
liuhaoyu@thinker:~/work/linux_driver/my_examples/completion$
```

更多 0

[上一篇](#) LDD3源码分析之调试技术

[下一篇](#) LDD3源码分析之ioctl操作

主题推荐

[源码](#)

[并发](#)

[驱动开发](#)

[并行处理](#)

[semaphore](#)

[猜你在找](#)

免费学习IT4个月,月薪12000

中国[官方授权]IT培训与就业示范基地,学成后名企直接招聘,月薪12000起!

[查看评论](#)

暂无评论

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

[全部主题](#) [Java](#) [VPN](#) [Android](#) [iOS](#) [ERP](#) [IE10](#) [Eclipse](#) [CRM](#) [JavaScript](#) [Ubuntu](#) [NFC](#)
[WAP](#) [jQuery](#) [数据库](#) [BI](#) [HTML5](#) [Spring](#) [Apache](#) [Hadoop](#) [.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](#) [Spark](#) [HBase](#) [Pure](#) [Solr](#) [Angular](#) [Cloud Foundry](#) [Redis](#) [Scala](#) [Django](#)
[Bootstrap](#)

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [银行汇款帐号](#) | [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](tel:400-600-2320)

京 ICP 证 070598 号

北京创新乐知信息技术有限公司 版权所有

江苏乐知网络技术有限公司 提供商务支持

Copyright © 1999-2014, CSDN.NET, All Rights Reserved 