

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: 80625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: 29970

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slab高速缓存

分类: LDD3源码分析

2012-03-31 14:07

1701人阅读

评论(4)

收藏

举报

cache

struct

destructor

数据结构

linux内核

constructor

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/scullc

本文分析LDD3第8章中关于使用slab高速缓存的代码，对应的源码在scullc目录下。另外，在较新的内核下编译scullc时会遇到一些错误，本文最后给出了解决这些错误的方法。

一、scullc源码分析

首先介绍一下slab相关的概念和函数。我们编写程序时，如果经常为某种数据结构进行分配和释放内存空间的操作，常常会用到空闲链表，其中包含多个已经分配好的可供使用的数据结构内存块，当需要使用数据结构时，直接去链表中去取，然后把数据放进去；使用完后，再把数据结构放回空闲链表，以供以后使用。

Linux内核中也有类似的需求，但是空闲链表的问题是不能全局控制，当内存紧缺时，内核无法通知每个空闲链表，让其释放出一些内存空间来。为此，Linux内核提供了slab分配器。简单理解，slab就是内核维护的一组大小不同的空闲链表，Linux把每个空闲链表称为后备高速缓存(lookaside cache)，当需要使用时，可以从中取出需要的内存块，不用时，再把内存块归还给slab。

slab维护的后备高速缓存是kmem_cache_t类型，可以由kmem_cache_create函数创建：

[cpp]

01. kmem_cache_t *kmem_cache_create(const char *name, size_t size,

02. size_t offset,

03. unsigned long flags,

04. void (*constructor)(void *, kmem_cache_t *,

05. unsigned long flags),

06. void (*destructor)(void *, kmem_cache_t *,

07. unsigned long flags));

kmem_cache_create函数创建一个新的后备高速缓存，其中可以容纳任意数目的内存块，这些内存块的大小都相同，由size参数指定。

使用kmem_cache_create创建了某个对象的后备高速缓存后，就可以调用kmem_cache_alloc从中分配内存对象：

<http://blog.csdn.net/liuhaoyutz/article/details/7415466>

1/8

最新评论

- LDD3源码分析之内存映射
wzw88486969:
@jhlhlong:unsigned long offset
= vma->vm_pgoff <v...
- Linux设备驱动程序架构分析之l2
teamos:看了你的l2c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
elecfan2011: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:jitc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
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
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kob (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! (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)

[cpp]

01. void *kmem_cache_alloc(kmem_cache_t *cache, int flags);

[cpp]

01. void kmem_cache_free(kmem_cache_t *cache, const void *obj);

释放一个内存对象使用kmem_cache_free函数:

如果驱动程序不会再使用后备高速缓存了,例如模块被卸载时,应该调用kmem_cache_destroy函数释
放后备高速缓存:

[cpp]

01. int kmem_cache_destroy(kmem_cache_t *cache);

我们知道了slab的相关概念和操作函数,下面可以看scullc的代码了。

scullc和前面分析过的scull绝大部分代码都是相同的,他们的区别在内存分配上,scullc使用slab
分配内存,而scull使用kmallo。下面我们只分析scullc与scull不同的代码,其他代码如果有问
题,大家可以参考前面分析scull的相关文章。

首先看scullc的模块初始化函数scullc_init,其中需要分析的是如下语句:

[cpp]

01. 560 scullc_cache = kmem_cache_create("scullc", scullc_quantum,
02. 561 0, SLAB_HWCACHE_ALIGN, NULL, NULL); /* no ctor/dtor */
03. 562 if (!scullc_cache) {
04. 563 scullc_cleanup();
05. 564 return -ENOMEM;
06. 565 }

scullc_cache定义在52行:

[cpp]

01. 51/* declare one cache pointer: use it for all devices */
02. 52kmem_cache_t *scullc_cache;

可以看出,在scullc的模块初始化函数中,创建了一个slab后备高速缓存,其关联的名称叫scullc,
其中包含的内存块的大小是scullc_quantum(默认值4000),每个内存块即是驱动程序中的一个量子。

现在有了后备高速缓存,下面scullc可以从为中为量子分配内存块了,相关代码在scullc_write函数
中,其中有必要分析的是如下语句:

[cpp]

01. 245 /* Allocate a quantum using the memory cache */
02. 246 if (!dptr->data[s_pos]) {
03. 247 dptr->data[s_pos] = kmem_cache_alloc(scullc_cache, GFP_KERNEL);
04. 248 if (!dptr->data[s_pos])
05. 249 goto nomem;
06. 250 memset(dptr->data[s_pos], 0, scullc_quantum);
07. 251 }

247行,从scullc_cache指向的后备高速缓存中分配了一个量子内存块。

scullc不再使用量子内存块时,应该返回给后备高速缓存,完成这项工作的函数是scullc_trim,其
中关键的代码如下:

[cpp]

01. 488 for (i = 0; i < qset; i++)
02. 489 if (dptr->data[i])
03. 490 kmem_cache_free(scullc_cache, dptr->data[i]);

最后,scullc模块卸载时,必须把后备高速缓存返回给系统,在scullc_cleanup函数中,有如下语

2014年01月 (1)

2013年12月 (6)

展开

文章搜索

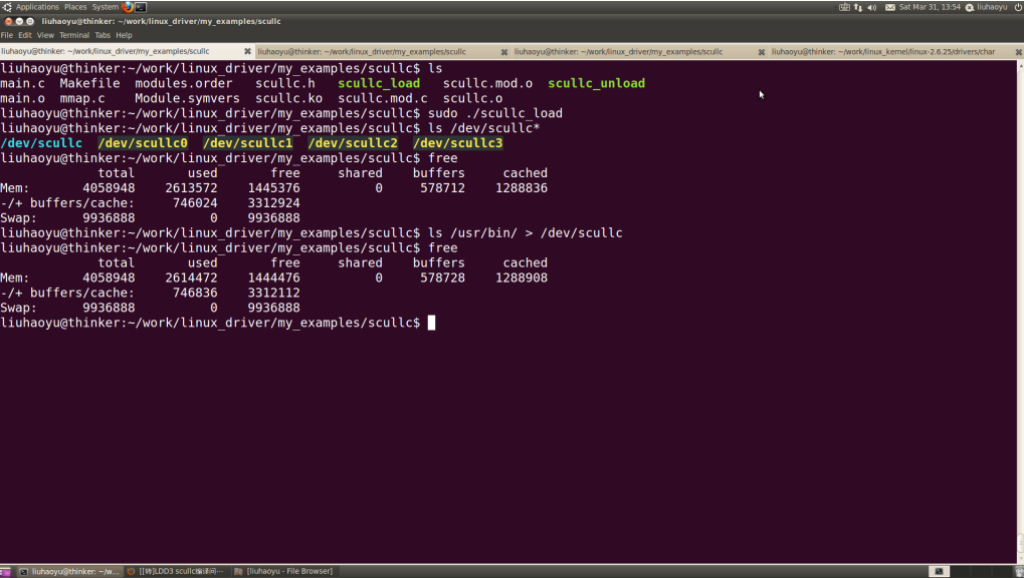
推荐文章

句:

```
[cpp]
01. 593     if (scullic_cache)
02. 594         kmem_cache_destroy(scullic_cache);
```

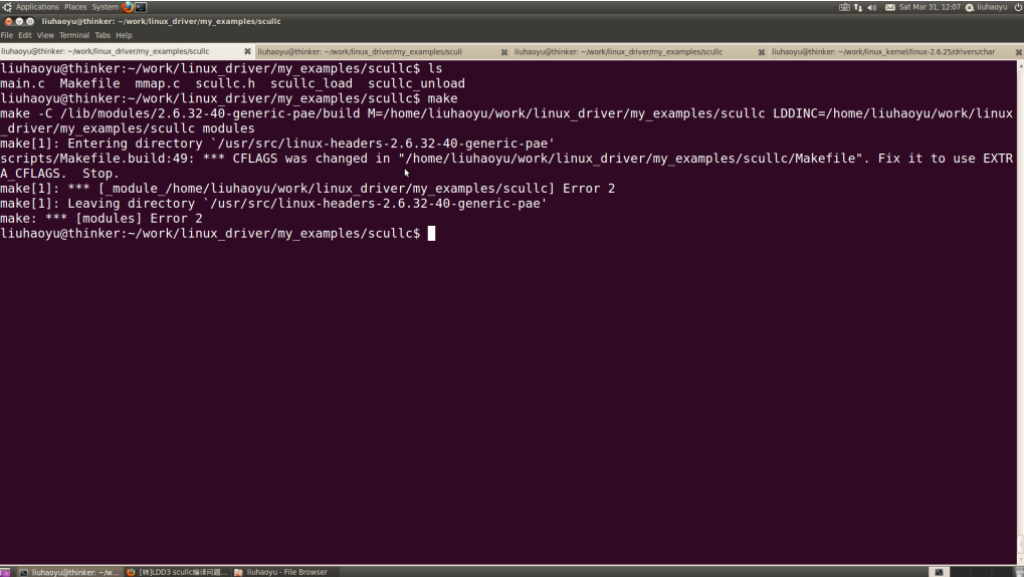
与scull相比，scullic的最大区别是运行速度略有提高，对内存利用率更高。由于后备调整缓存中的内存块都是同样的大小，所以其在内存中的排列位置达到了最大密集程度，相反，scull的数据对象则会引起不可预测的内存碎片。

在我的机器上，测试scullic模块过程如下图所示：

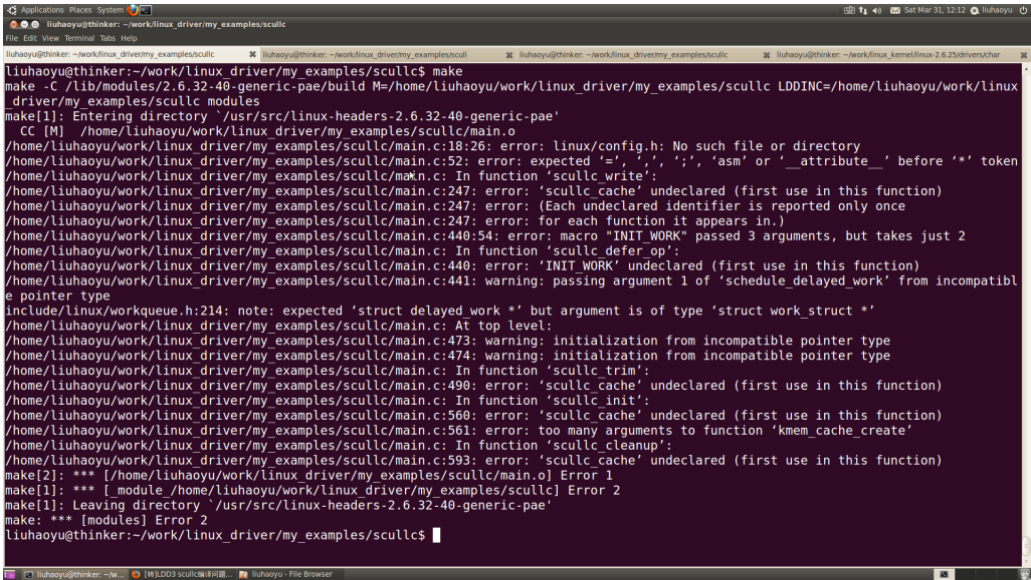


二、编译scullic时遇到的问题

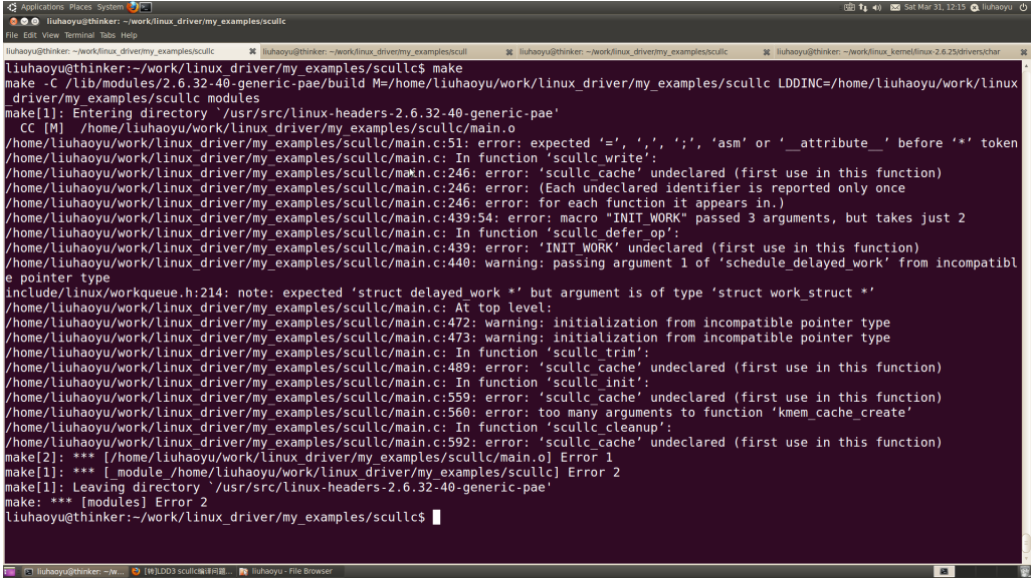
在2.6.32-38-generic-pae上编译LDD3提供的scullic模块时，会出现如下错误：



把Makefile中第12行和第42行的CFLAGS替换为EXTRA_CFLAGS即可解决上面的错误。再次make，会出现如下错误：



因为linux/config.h现在已经不存在了，所以直接把main.c的第18行去掉，即可解决这个错误，再次编译，出现如下错误信息：

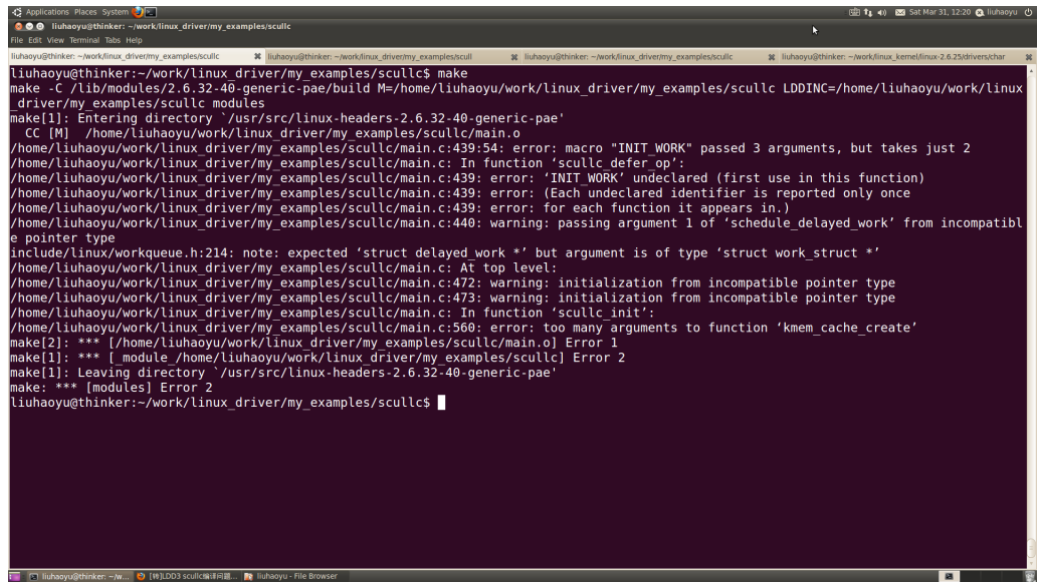


这是因为新内核中不再使用kmem_cache_t为个类型定义，而是使用struct kmem_cache结构体代表一个后备高速缓存。所以把main.c的第51行改为

[cpp]

```
01. struct kmem_cache *scullc_cache;
```

即可解决这个问题，再次make，又出现如下错误：



```
liuhaoyu@thinker:~/work/linux_driver/my_examples/scullc$ make
make -C /lib/modules/2.6.32-40-generic-pae/build M=/home/liuhaoyu/work/linux_driver/my_examples/scullc LDDINC=/home/liuhaoyu/work/linux_driver/my_examples/scullc modules
make[1]: Entering directory '/usr/src/linux-headers-2.6.32-40-generic-pae'
  CC [M] /home/liuhaoyu/work/linux_driver/my_examples/scullc/main.o
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:439:54: error: macro "INIT_WORK" passed 3 arguments, but takes just 2
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c: In function 'scullc_defer_op':
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:439: error: 'INIT_WORK' undeclared (first use in this function)
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:439: error: (Each undeclared identifier is reported only once
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:439: error: for each function it appears in.)
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:440: warning: passing argument 1 of 'schedule_delayed_work' from incompatible pointer type
include/linux/workqueue.h:214: note: expected 'struct delayed_work *' but argument is of type 'struct work_struct *'
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c: At top level:
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:472: warning: initialization from incompatible pointer type
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:473: warning: initialization from incompatible pointer type
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c: In function 'scullc_init':
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:560: error: too many arguments to function 'kmem_cache_create'
make[2]: *** [/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.o] Error 1
make[1]: *** [module /home/liuhaoyu/work/linux_driver/my_examples/scullc] Error 2
make[1]: Leaving directory '/usr/src/linux-headers-2.6.32-40-generic-pae'
make: *** [modules] Error 2
liuhaoyu@thinker:~/work/linux_driver/my_examples/scullc$
```

这是因为在新内核中INIT_WORK宏发生了变化，现在INIT_WORK只能接受两个参数，所以将439行改为：

[cpp]

```
01. 439 INIT_WORK(&stuff->work, scullc_do_deferred_op);
```

注意，新的INIT_WORK的第二个参数scullc_do_deferred_op函数以INIT_WORK的第一个参数做为参数。

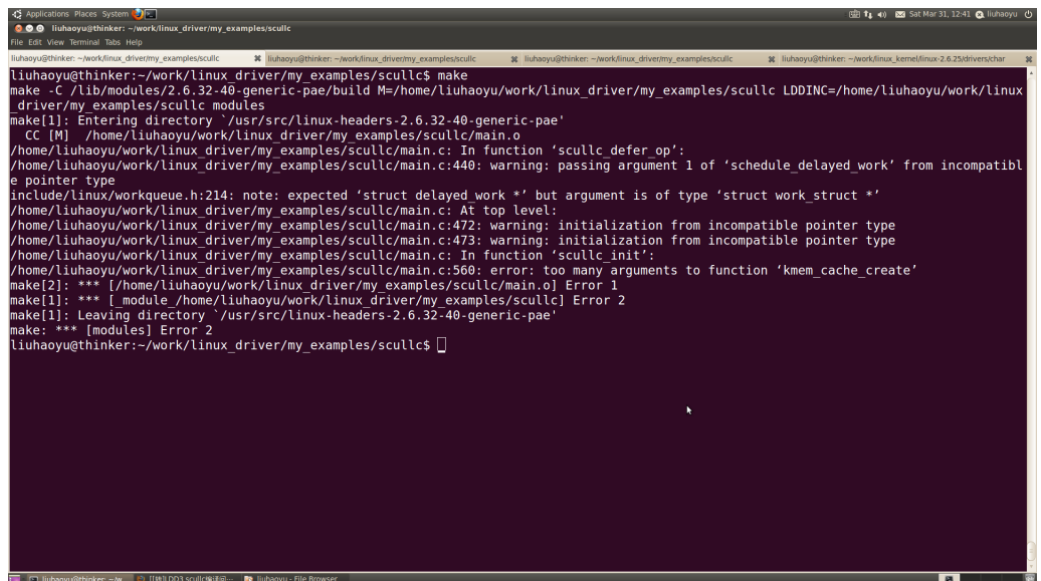
连锁反应，还需要把scullc_do_deferred_op函数的实现修改如下：

[cpp]

```
01. 409static void scullc_do_deferred_op(struct work_struct *p)
02. 410{
03. 411 struct async_work *stuff = container_of(p, struct async_work, work);
04. 412 aio_complete(stuff->iocb, stuff->result, 0);
05. 413 kfree(stuff);
06. 414}
```

409行和411行发生了变化。

再次编译，出现如下错误：



```
liuhaoyu@thinker:~/work/linux_driver/my_examples/scullc$ make
make -C /lib/modules/2.6.32-40-generic-pae/build M=/home/liuhaoyu/work/linux_driver/my_examples/scullc LDDINC=/home/liuhaoyu/work/linux_driver/my_examples/scullc modules
make[1]: Entering directory '/usr/src/linux-headers-2.6.32-40-generic-pae'
  CC [M] /home/liuhaoyu/work/linux_driver/my_examples/scullc/main.o
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:440: warning: passing argument 1 of 'schedule_delayed_work' from incompatible pointer type
include/linux/workqueue.h:214: note: expected 'struct delayed_work *' but argument is of type 'struct work_struct *'
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c: At top level:
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:472: warning: initialization from incompatible pointer type
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:473: warning: initialization from incompatible pointer type
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c: In function 'scullc_init':
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:560: error: too many arguments to function 'kmem_cache_create'
make[2]: *** [/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.o] Error 1
make[1]: *** [module /home/liuhaoyu/work/linux_driver/my_examples/scullc] Error 2
make[1]: Leaving directory '/usr/src/linux-headers-2.6.32-40-generic-pae'
make: *** [modules] Error 2
liuhaoyu@thinker:~/work/linux_driver/my_examples/scullc$
```

这是因为，在新内核中，kmem_cache_create函数发生了变化，最后一个参数destructor被删除掉了。所以，把560行最后一个参数NULL删除即可解决这个问题。再次编译，编译通过，但是还有几个警告信息，如下图所示：


```

liuhaoyu@thinker:~/work/linux_driver/my_examples/scullc$ ls
main.c  Makefile  mmap.c  scullc.h  scullc.load  scullc.unload
liuhaoyu@thinker:~/work/linux_driver/my_examples/scullc$ make
make -C /lib/modules/2.6.32-40-generic-pae/build M=/home/liuhaoyu/work/linux_driver/my_examples/scullc LDDINC=/home/liuhaoyu/work/linux_driver/my_examples/scullc modules
make[1]: Entering directory '/usr/src/linux-headers-2.6.32-40-generic-pae'
  CC [M] /home/liuhaoyu/work/linux_driver/my_examples/scullc/main.o
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c: In function 'scullc_defer_op':
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:440: warning: passing argument 1 of 'schedule_delayed_work' from incompatible pointer type
include/linux/workqueue.h:214: note: expected 'struct delayed_work *' but argument is of type 'struct work_struct *'
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c: At top level:
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:472: warning: initialization from incompatible pointer type
/home/liuhaoyu/work/linux_driver/my_examples/scullc/main.c:473: warning: initialization from incompatible pointer type
  LD [M] /home/liuhaoyu/work/linux_driver/my_examples/scullc/scullc.o
Building modules, stage 2.
MODPOST 1 modules
  CC      /home/liuhaoyu/work/linux_driver/my_examples/scullc/scullc.mod.o
  LD [M] /home/liuhaoyu/work/linux_driver/my_examples/scullc/scullc.ko
make[1]: Leaving directory '/usr/src/linux-headers-2.6.32-40-generic-pae'
liuhaoyu@thinker:~/work/linux_driver/my_examples/scullc$

```

这是因为，在2.6.32-38-generic-pae内核中，schedule_delayed_work函数的定义发生了变化，现在其函数原型是

[cpp]

```
01. int schedule_delayed_work(struct delayed_work *dwork, unsigned long delay)
```

而在LDD3使用的2.6.10版本的内核中，其函数原型是：

[cpp]

```
01. int fastcall schedule_delayed_work(struct work_struct *work, unsigned long delay)
```

所以要在新的内核上执行schedule_delayed_work，原来的work_struct必须改为delayed_work。

delayed_work结构体定义如下：

[cpp]

```

01. struct delayed_work {
02.     struct work_struct work;
03.     struct timer_list timer;
04. };

```

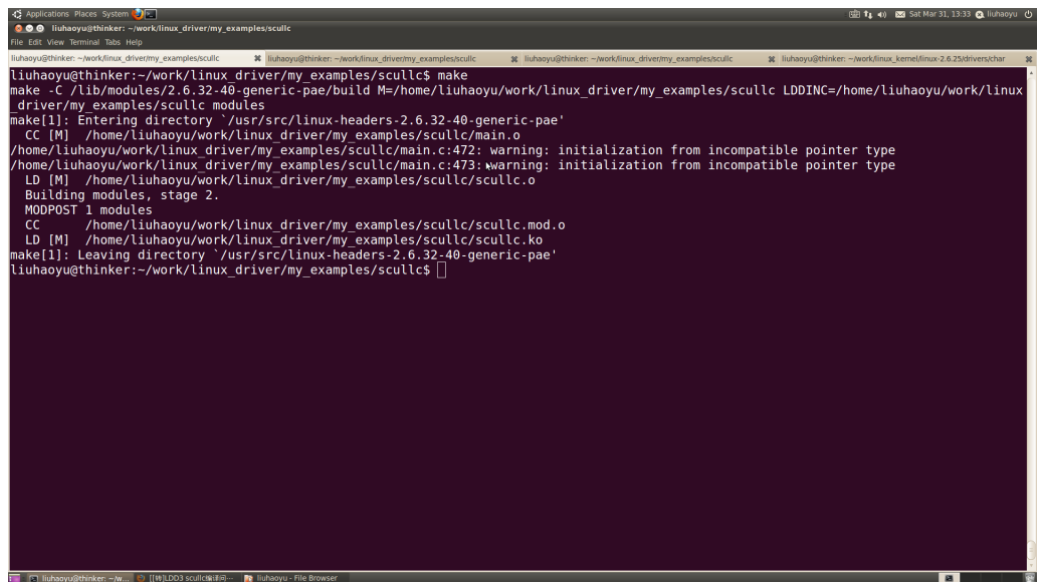
为了修正这个警告，需要修改如下三个地方：

403行改为： struct delayed_work work;

411行改为： struct async_work *stuff = container_of(p, struct async_work, work.work);

439行改为： INIT_DELAYED_WORK(&stuff->work, scullc_do_deferred_op);

再次编译，还有如下警告信息：



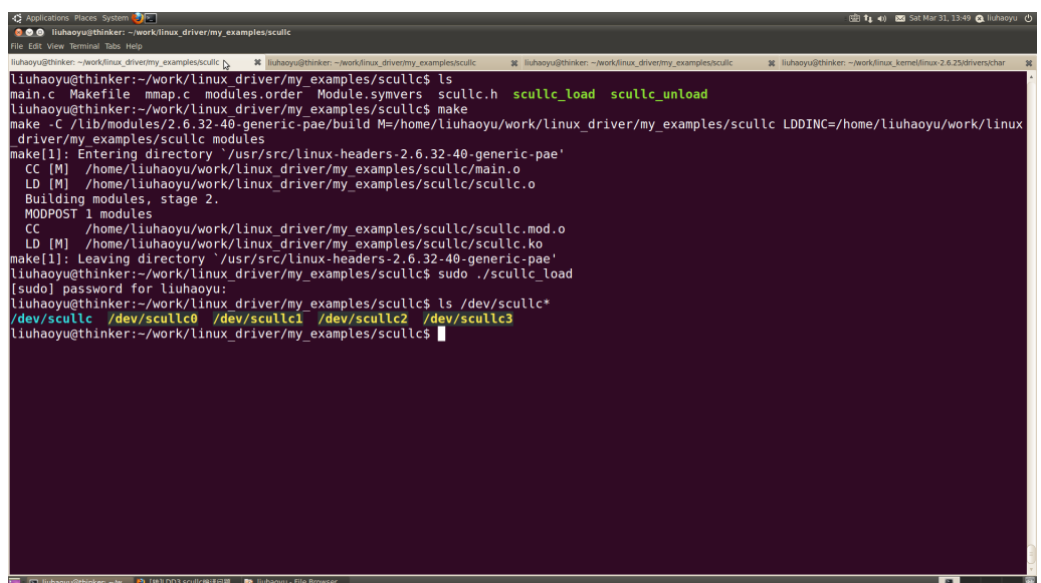
这是因为在新内核中，file_operations结构体的aio_read和aio_write成员函数原型发生了变化。所以做如下修改：

```
[cpp]
01. 445static ssize_t scullc_aio_read(struct kiocb *iocb, const struct iovec *buf, unsigned long
02.                                count,
03.                                loff_t pos)
04. 447{
05. 448     return scullc_defer_op(0, iocb, (char __user *) buf, count, pos);
06. 449}
07. 450
08. 451static ssize_t scullc_aio_write(struct kiocb *iocb, const struct iovec *buf,
09.                                unsigned long count, loff_t pos)
10. 453{
11. 454     return scullc_defer_op(1, iocb, (char __user *) buf, count, pos);
12. 455}
```

改动的地方是445行和451行两个函数的函数原型，使参数类型符合新的定义。

另外还修改了448行，把第三个参数强制转换为char user*类型。

修改后，编译成功，如下图所示：



更多 0

上一篇 [LDD3源码分析之时间与延迟操作](#)

下一篇 [LDD3源码分析之按页分配内存](#)

顶

0

踩

0

主题推荐

源码

linux内核

数据结构

内存分配

对象

猜你在找

- Vibrator Kernel driver 实现

Linux设备驱动开发详解-Note(16)---Linux 设备驱动中
- Linux下串口相关的几个有用的命令

无序hashset与hashmap让其有序
- Android 用Vibrator实现震动功能

如何在windows下面编译u-boot （原发于：2012-07-24
- 内核符号表问题

获取Linux 内存页大小的命令
- u-boot编译笔记


Linux PPP 数据收发流程

免 费 学 习 IT 4 个 月 , 月 薪 1 2 0 0 0

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

4楼 [donghuwuwei](#) 2014-02-15 22:43发表




省去了不少修改的时间，真是太好了

3楼 [捧灰](#) 2013-10-27 19:51发表



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-xen-i686

2楼 [sunstars2009918](#) 2012-12-03 22:43发表



专业级水准，学习不少，多谢

1楼 [liuyang19890710](#) 2012-07-11 08:48发表



写的很不错，大神级别的。自己吸收重新组织了，不错不错。。。

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

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