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

LDD3源码分析之异步通知

分类： LDD3源码分析 2012-03-28 09:09 1137人阅读 评论(0) 收藏 举报

struct asynchronous 数据结构 semaphore action buffer

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/pipe.c examples/scull/main.c

一、异步通知机制的实现

本文分析LDD3第6章中的异步通知机制。

通过使用异步通知机制，应用程序可以在指定的I/O操作可执行时，收到一个信号，而不需要不停的使用轮询来查询设备。

要使用异步通知机制，对于用户空间程序来说，需要执行如下步骤：

首先，指定一个进程作为文件的“属主”。这是通过使用fcntl系统调用执行F_SETOWN命令完成的，该命令会把进程ID号保存在filp->f_owner中。这一步的目的是让内核知道应该通知哪个进程。

其次，在设备中设置FASYNC标志，这是通过fcntl的F_SETFL命令完成的。

执行完这两个步骤后，当指定的I/O操作可执行时，就会给相应进程发送一个SIGIO信号。

从内核的角度看，要实现异步通知机制，需要经过如下三个步骤：

首先，F_SETOWN被调用时，对filp->f_owner赋值，此外什么也不做。

其次，执行F_SETFL设置FASYNC时，调用驱动程序的fasync函数。只要filp->f_flags中的FASYNC标志发生了变化，就应该调用这个函数，以便把这个变化通知驱动程序，使其能做出正确响应。文件打开时，FASYNC标志默认是被清除的。

第三，当指定的I/O操作可执行时，所有注册为异步通知的进程都会被发送一个SIGIO信号。

第一步的实现很简单，在驱动程序部分没有什么可做的。第二步和第三部则要涉及维护一个动态数据结构，以跟踪不同的异步读取进程，这种进程可能有好几个。不过，这个动态数据结构并不依赖于特定设备，内核已经提供了一套通用的实现方法，没有必要为每个驱动程序重写这部分代码。

Linux的这种通用方法基于一个数据结构和两个函数，这个数据结构是struct fasync_struct，该结构体在linux/fs.h中定义如下：

```
[cpp]
01. struct fasync_struct {
```

```
02.     int magic;
03.     int fa_fd;
04.     struct  fasync_struct   *fa_next; /* singly linked list */
05.     struct  file        *fa_file;
06. };
```

和处理等待队列的方式类似，我们需要把一个该类型的指针插入设备特定的数据结构中去。回忆一下scullpipe设备结构体的定义：

```
[cpp]

01. 33struct scull_pipe {
02. 34        wait_queue_head_t inq, outq;       /* read and write queues */
03. 35        char *buffer, *end;                /* begin of buf, end of buf */
04. 36        int buffersize;                    /* used in pointer arithmetic */
05. 37        char *rp, *wp;                     /* where to read, where to write */
06. 38        int nreaders, nwriters;            /* number of openings for r/w */
07. 39        struct fasync_struct *async_queue; /* asynchronous readers */
08. 40        struct semaphore sem;              /* mutual exclusion semaphore */
09. 41        struct cdev cdev;                  /* Char device structure */
10. 42};
```

第39行定义了fasync_struct结构体变量。

两个相关函数分别是：

```
[cpp]

01. int fasync_helper(int fd, struct file * filp,
02.                   int on, struct fasync_struct **fapp);
03. void kill_fasync(struct fasync_struct **fp, int sig, int band)
```

当一个打开的文件的FASYNC标志被修改时，调用fasync_helper以便从相关的进程列表中增加或删除文件。除了最后一个参数外，fasync_helper的其它参数与驱动程序的fasync函数相同，可以直接传递。

scullpipe设备的fasync函数实现如下：

```
[cpp]

01. 253static int scull_p_fasync(int fd, struct file *filp, int mode)
02. 254{
03. 255    struct scull_pipe *dev = filp->private_data;
04. 256
05. 257    return fasync_helper(fd, filp, mode, &dev->async_queue);
06. 258}
```

当指定的I/O操作可执行时，应使用kill_fasync通知所有的相关进程，该函数的第二个参数是要发送的信号(通常是SIGIO)，第三个参数是带宽(通常是POLL_IN)。由于提供给scullpipe的读取进程的新数据是由某个进程调用write产生的，所以kill_fasync函数在scullpipe的write函数中调用，代码片段如下所示：

```
[cpp]

01. 221    /* and signal asynchronous readers, explained late in chapter 5 */
02. 222    if (dev->async_queue)
03. 223        kill_fasync(&dev->async_queue, SIGIO, POLL_IN);
```

如果是针对写入的异步通知，kill_fasync的第三个参数必须为POLL_OUT。

最后要做的是，当文件关闭时，必须调用fasync方法，以便从活动的异步通知进程列表中删除该文件。在scullpipe的close函数中，有如下代码：

```
[cpp]

01. 98    /* remove this filp from the asynchronously notified filp's */
```

2014年01月 (1)
2013年12月 (6)
展开

文章搜索

推荐文章

```
02.  99    scull_p_fasync(-1, filp, 0);
```

二、测试scullpipe的异步通知机制

LDD3提供了一个异步通知机制的测试程序examples/misc-progs/asynctest.c，其代码如下：

```
[cpp]

01.  /*
02.   * asynctest.c: use async notification to read stdin
03.   *
04.   * Copyright (C) 2001 Alessandro Rubini and Jonathan Corbet
05.   * Copyright (C) 2001 O'Reilly & Associates
06.   *
07.   * The source code in this file can be freely used, adapted,
08.   * and redistributed in source or binary form, so long as an
09.   * acknowledgment appears in derived source files.  The citation
10.  * should list that the code comes from the book "Linux Device
11.  * Drivers" by Alessandro Rubini and Jonathan Corbet, published
12.  * by O'Reilly & Associates.   No warranty is attached;
13.  * we cannot take responsibility for errors or fitness for use.
14.  */
15.
16.  #include <stdio.h>
17.  #include <stdlib.h>
18.  #include <string.h>
19.  #include <unistd.h>
20.  #include <signal.h>
21.  #include <fcntl.h>
22.
23.  int gotdata=0;
24.  void sighandler(int signo)
25.  {
26.      if (signo==SIGIO)
27.          gotdata++;
28.      return;
29.  }
30.
31.  char buffer[4096];
32.
33.  int main(int argc, char **argv)
34.  {
35.      int count;
36.      struct sigaction action;
37.
38.      memset(&action, 0, sizeof(action));
39.      action.sa_handler = sighandler;
40.      action.sa_flags = 0;
41.
42.      sigaction(SIGIO, &action, NULL);
43.
44.      fcntl(STDIN_FILENO, F_SETOWN, getpid());
45.      fcntl(STDIN_FILENO, F_SETFL, fcntl(STDIN_FILENO, F_GETFL) | FASYNC);
46.
47.      while(1) {
48.          /* this only returns if a signal arrives */
49.          sleep(86400); /* one day */
50.          if (!gotdata)
51.              continue;
52.          count=read(0, buffer, 4096);
53.          /* buggy: if avail data is more than 4kbytes... */
54.          write(1,buffer,count);
55.          gotdata=0;
56.      }
57.  }
```

第38 – 42行，设置信号处理函数sighandler。

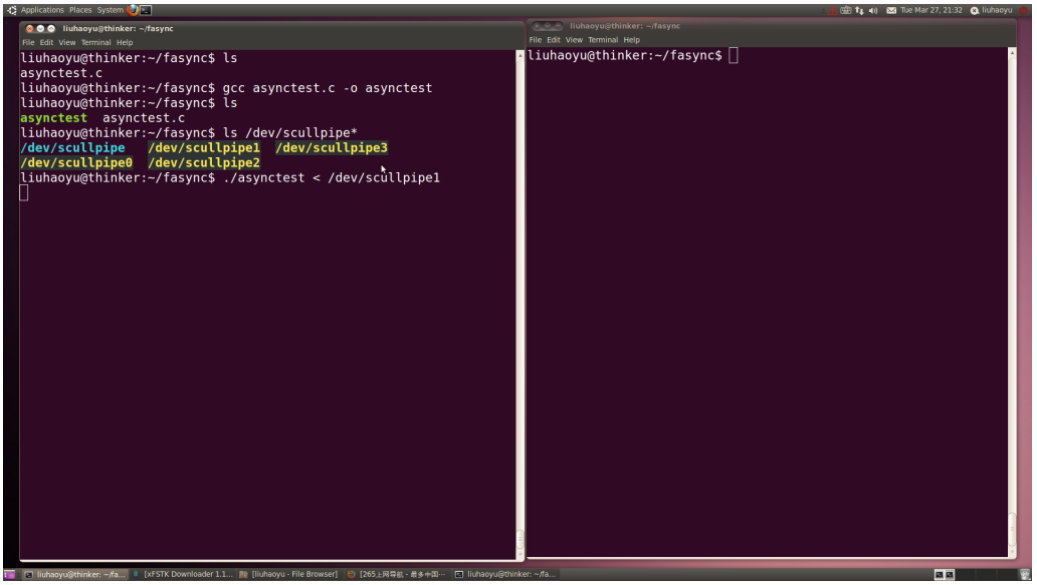
第24 – 29行，是信号处理函数sighandler的实现。

第44行，指定当前进程作为标准输入设备的“属主”。

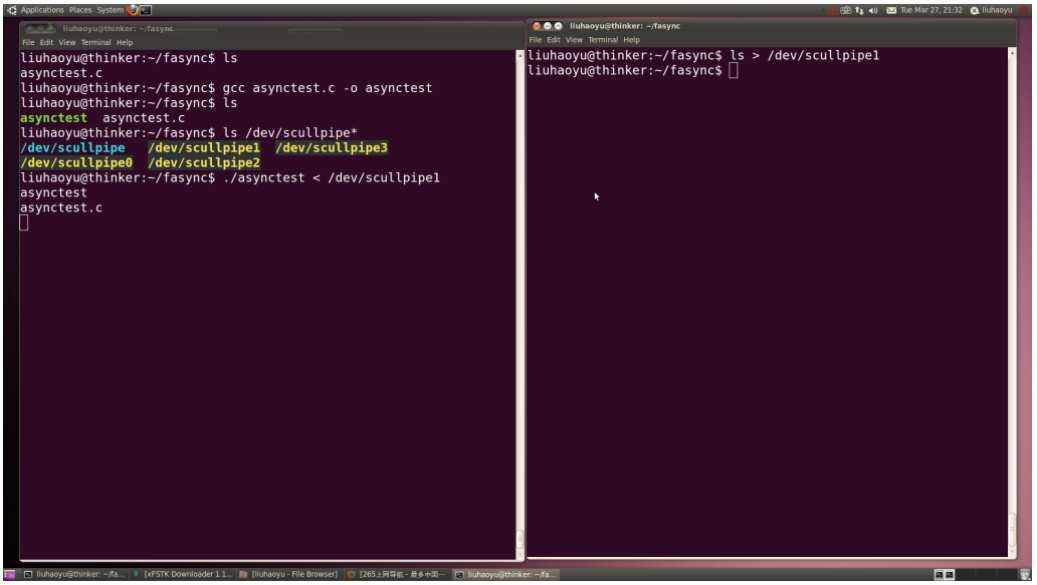
第45行，在标准输入设备中设置FASYNC标志。

第47 – 56行，循环等待，当输入设备有数据可读时，会发信号唤醒进程，并读取和打印信息。

使用该程序测试scullpipe的过程如下图所示：



我们把asynctest的标准输入重定向到/dev/scullpipe1，所以当/dev/scullpipe1有数据可读，时会向asynctest发信号SIGIO，唤醒asynctest，执行信号处理函数，然后读取并打印信息，再进入下次循环。如下图所示：



更多 0

上一篇 LDD3源码分析之poll分析

下一篇 LDD3源码分析之llseek分析

主题推荐 源码 异步 数据结构 asynchronous 应用程序

猜你在找

LDD3源码分析之slab高速缓存
Touch Driver介绍
Ubuntu下JNI的实现与调用
select()和poll()的区别是什么？
u-boot编译笔记
DDR，DDR2與DDR3的區別
Android 用Vibrator实现震动功能
linux线程初探（转载）
如何在windows下面编译u-boot （原发于：2012-07-24
STM32 对内部FLASH读写接口函数

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