

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应用开发学习笔记本

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poll分析

分类: LDD3源码分析

2012-03-27 18:43

1972人阅读

评论(2)

收藏

举报

struct

table

测试

events

数据结构

descriptor

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

本文分析LDD3第6章的poll(轮询)操作。要理解驱动程序中poll函数的作用和实现，必须先理解用户空间中poll和select函数的用法。本文与前面的文章介绍的顺序有所不同，首先分析测试程序，以此理解用户空间中的poll和select函数的用法。然后再分析驱动程序怎样对用户空间的poll和select函数提供支持。

一、poll函数的使用

用户态的poll函数用以监测一组文件描述符是否可以执行指定的I/O操作，如果被监测的文件描述符都不能执行指定的I/O操作，则poll函数会阻塞，直到有文件描述符的状态发生变化，可以执行指定的I/O操作才解除阻塞。poll函数还可以指定一个最长阻塞时间，如果时间超时，将直接返回。poll函数的函数原型如下：

```
[cpp]
01. int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

要监测的文件描述符由第一个参数fds指定，它是一个struct pollfd数组，pollfd结构体定义如下：

```
[cpp]
01. struct pollfd {
02.     int fd;           /* file descriptor */
03.     short events;      /* requested events */
04.     short revents;     /* returned events */
05. };
```

pollfd结构体的第一个成员fd是文件描述符，代表一个打开的文件。第二个成员events是一个输入参数，用于指定poll监测哪些事件(如可读、可写等等)。第三个成员revents是一个输出参数，由内核填充，指示对于文件描述符fd，发生了哪些事件(如可读、可写等等)。

poll函数的第二个参数nfds代表监测的文件描述符的个数，即fds数组的成员个数。

poll函数的第三个参数timeout代表阻塞时间(以毫秒为单位)，如果poll要求监测的事件没有发生，则poll会阻塞最

<http://blog.csdn.net/liuhaoyutz/article/details/7400037>

1/10

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
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: 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
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ADI (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)

多timeout毫秒。如果timeout设置为负数，则poll会一直阻塞，直到监测的事件发生。

poll函数如果返回一个正数，代表内核返回了状态(保存在pollfd.revents中)的文件描述符的个数。如果poll返回0，表明是因为超时而返回的。如果poll返回-1，表明poll调用出错。

poll函数可以监测哪些状态(由pollfd.events指定)，以及内核可以返回哪些状态(保存在pollfd.revents中)，由下面的宏定义：

POLLIN: There is data to read.

POLLOUT: Writing now will not block.

POLLPRI: There is urgent data to read (e.g., out-of-band data on TCP socket; pseudo-terminal master in packet mode has seen state change in slave).

POLLRDHUP: (since Linux 2.6.17) Stream socket peer closed connection, or shut down writing half of connection. The _GNU_SOURCE feature test macro must be defined in order to obtain this definition.

POLLERR: Error condition (output only).

POLLHUP: Hang up (output only).

POLLNVAL: Invalid request: fd not open (output only).

When compiling with _XOPEN_SOURCE defined, one also has the following, which convey no further information beyond the bits listed above:

POLLRDNORM: Equivalent to POLLIN.

POLLRDBAND: Priority band data can be read (generally unused on Linux).

POLLWRNORM: Equivalent to POLLOUT.

POLLWRBAND: Priority data may be written.

下面我们看一个测试scullpipe设备的poll操作(内核态poll)的测试程序，该程序使用了我们前面介绍的poll函数(用户态poll)。其代码如下：

```
[cpp]
01. #include <stdio.h>
02. #include <stdlib.h>
03. #include <string.h>
04. #include <fcntl.h>
05. #include <unistd.h>
06. #include <linux/poll.h>
07. #include <sys/time.h>
08. #include <sys/types.h>
09. #include <sys/stat.h>
10.
11. int main(int argc, char *argv[])
12. {
13.     int fd0, fd1, fd2, fd3;
14.     struct pollfd poll_fd[4];
15.     char buf[100];
16.     int retval;
17.
18.     if(argc != 2 || ((strcmp(argv[1], "read") != 0) && (strcmp(argv[1], "write") != 0)))
19.     {
20.         printf("usage: ./poll_test read|write\n");
21.         return -1;
22.     }
23.
24.     fd0 = open("/dev/scullpipe0", O_RDWR);
25.     if ( fd0 < 0)
26.     {
27.         printf("open scullpipe0 error\n");
28.         return -1;
29.     }
30.
31.     fd1 = open("/dev/scullpipe1", O_RDWR);
32.     if ( fd1 < 0)
```

[2014年01月 \(1\)](#)[2013年12月 \(6\)](#)[展开](#)

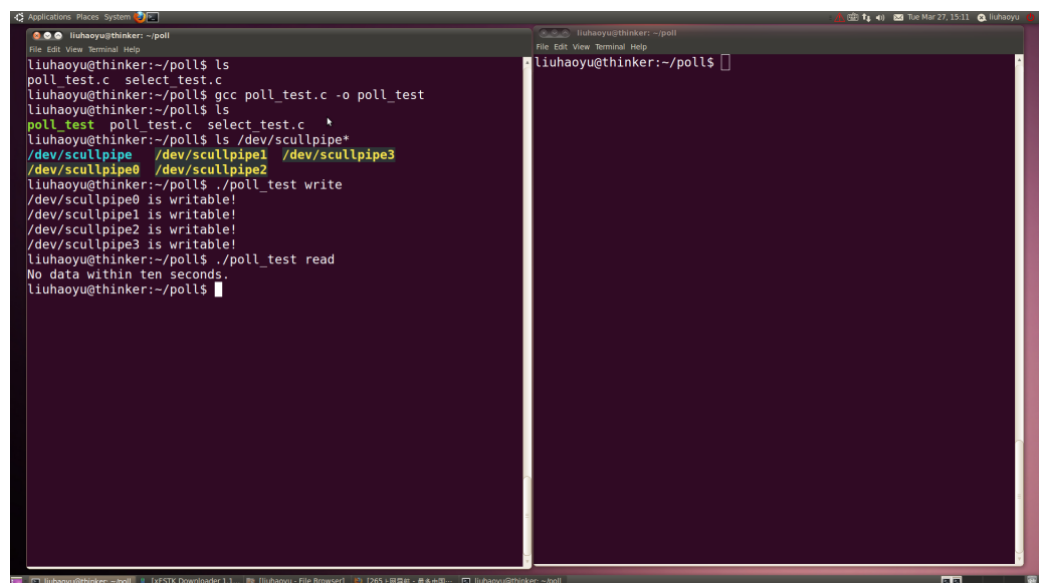
文章搜索

推荐文章

```
33. {
34.     printf("open scullpipe1 error\n");
35.     return -1;
36. }
37.
38. fd2 = open("/dev/scullpipe2", O_RDWR);
39. if ( fd2 < 0)
40. {
41.     printf("open scullpipe2 error\n");
42.     return -1;
43. }
44.
45. fd3 = open("/dev/scullpipe3", O_RDWR);
46. if ( fd3 < 0)
47. {
48.     printf("open scullpipe3 error\n");
49.     return -1;
50. }
51.
52. if(strcmp(argv[1], "read") == 0)
53. {
54.     poll_fd[0].fd = fd0;
55.     poll_fd[1].fd = fd1;
56.     poll_fd[2].fd = fd2;
57.     poll_fd[3].fd = fd3;
58.
59.     poll_fd[0].events = POLLIN | POLLRDNORM;
60.     poll_fd[1].events = POLLIN | POLLRDNORM;
61.     poll_fd[2].events = POLLIN | POLLRDNORM;
62.     poll_fd[3].events = POLLIN | POLLRDNORM;
63.
64.     retval = poll(poll_fd, 4, 10000);
65. }
66. else
67. {
68.     poll_fd[0].fd = fd0;
69.     poll_fd[1].fd = fd1;
70.     poll_fd[2].fd = fd2;
71.     poll_fd[3].fd = fd3;
72.
73.     poll_fd[0].events = POLLOUT | POLLWRNORM;
74.     poll_fd[1].events = POLLOUT | POLLWRNORM;
75.     poll_fd[2].events = POLLOUT | POLLWRNORM;
76.     poll_fd[3].events = POLLOUT | POLLWRNORM;
77.
78.     retval = poll(poll_fd, 4, 10000);
79. }
80.
81. if (retval == -1)
82. {
83.     printf("poll error!\n");
84.     return -1;
85. }
86. else if (retval)
87. {
88.     if(strcmp(argv[1], "read") == 0)
89.     {
90.         if(poll_fd[0].revents & (POLLIN | POLLRDNORM))
91.         {
92.             printf("/dev/scullpipe0 is readable!\n");
93.             memset(buf, 0, 100);
94.             read(fd0, buf, 100);
95.             printf("%s\n", buf);
96.         }
97.
98.         if(poll_fd[1].revents & (POLLIN | POLLRDNORM))
99.         {
100.             printf("/dev/scullpipe1 is readable!\n");
101.             memset(buf, 0, 100);
102.             read(fd1, buf, 100);
103.             printf("%s\n", buf);
104.         }
105.
106.         if(poll_fd[2].revents & (POLLIN | POLLRDNORM))
107.         {
108.             printf("/dev/scullpipe2 is readable!\n");
109.             memset(buf, 0, 100);
110.             read(fd2, buf, 100);
111.             printf("%s\n", buf);
```

```
112.         }
113.
114.         if(poll_fd[3].revents & (POLLIN | POLLRDNORM))
115.         {
116.             printf("/dev/scullpipe3 is readable!\n");
117.             memset(buf, 0, 100);
118.             read(fd3, buf, 100);
119.             printf("%s\n", buf);
120.         }
121.     }
122.     else
123.     {
124.         if(poll_fd[0].revents & (POLLOUT | POLLWRNORM))
125.         {
126.             printf("/dev/scullpipe0 is writable!\n");
127.         }
128.
129.         if(poll_fd[1].revents & (POLLOUT | POLLWRNORM))
130.         {
131.             printf("/dev/scullpipe1 is writable!\n");
132.         }
133.
134.         if(poll_fd[2].revents & (POLLOUT | POLLWRNORM))
135.         {
136.             printf("/dev/scullpipe2 is writable!\n");
137.         }
138.
139.         if(poll_fd[3].revents & (POLLOUT | POLLWRNORM))
140.         {
141.             printf("/dev/scullpipe3 is writable!\n");
142.         }
143.     }
144. }
145. else
146. {
147.     if(strcmp(argv[1], "read") == 0)
148.     {
149.         printf("No data within ten seconds.\n");
150.     }
151.     else
152.     {
153.         printf("Can not write within ten seconds.\n");
154.     }
155. }
156.
157. return 0;
158. }
```

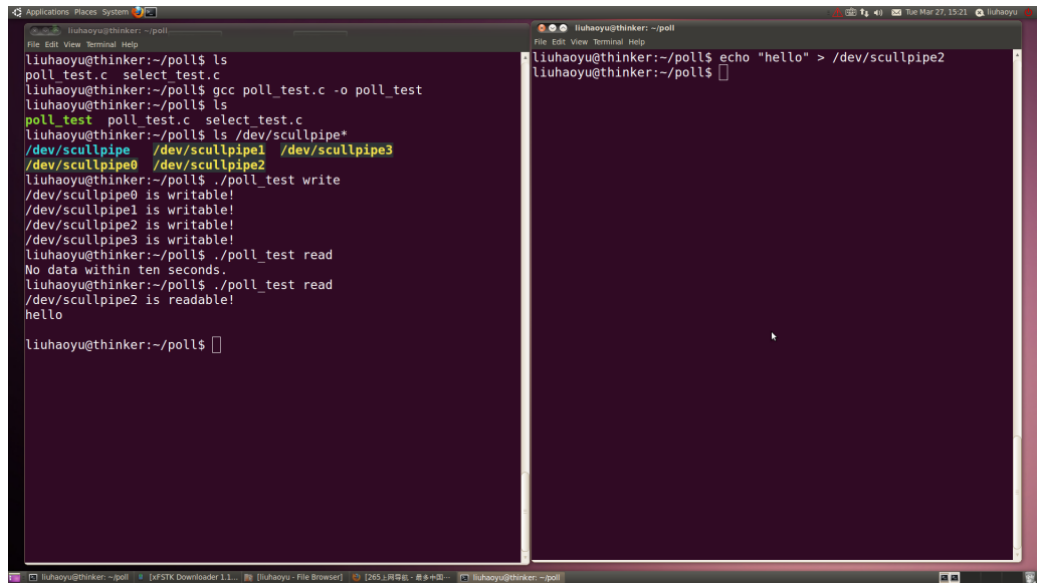
测试过程如下图所示:



```
liuhaoyu@thinker:~/poll$ ls
poll_test.c  select_test.c
liuhaoyu@thinker:~/poll$ gcc poll_test.c -o poll_test
liuhaoyu@thinker:~/poll$ ls
poll_test  poll_test.c  select_test.c
liuhaoyu@thinker:~/poll$ ls /dev/scullpipe*
/dev/scullpipe  /dev/scullpipe1  /dev/scullpipe3
/dev/scullpipe0  /dev/scullpipe2
liuhaoyu@thinker:~/poll$ ./poll_test write
/dev/scullpipe0 is writable!
/dev/scullpipe1 is writable!
/dev/scullpipe2 is writable!
/dev/scullpipe3 is writable!
liuhaoyu@thinker:~/poll$ ./poll_test read
No data within ten seconds.
liuhaoyu@thinker:~/poll$
```

从上图可以看出, scullpipe0 - scullpipe3都是可写的。但是因为没有向其中写入任何内容, 所以读的时候会阻塞住, 因为测试程序设置最长阻塞时间为10秒, 所以10秒后解除阻塞退出, 并打印“No data within ten seconds.”。

下面再次测试读操作，因为设备中没有内容，还是阻塞住，但是在10秒钟内，从另外一个终端向/dev/scullpipe2写入数据，这里是写入字符串“hello”，可以看到，写入数据后，测试程序解除阻塞，并打印“/dev/scullpipe2 is readable!”和“hello”字符串，这个过程如下图所示：



二、select函数的使用

select函数的作用与poll函数类似，也是监测一组文件描述符是否准备好执行指定的I/O操作。如果没有任何一个文件描述符可以完成指定的操作，select函数会阻塞住。select函数可以指定一个最长阻塞时间，如果超时，则直接返回。

select函数的函数原型如下：

```
[cpp]
01. int select(int nfds, fd_set *readfds, fd_set *writefds,
02.            fd_set *exceptfds, struct timeval *timeout);
```

select通过三个独立的文件描述符集(fd_set)readfds, writefds, exceptfds指示监测哪些文件描述符，其中readfds中的文件描述符监测是否可进行非阻塞的读操作。writefds数组中的文件描述符监测是否可进行非阻塞的写操作。exceptfds数组中的文件描述符监测是否有异常(exceptions)。

select的第一个参数nfds是三个文件描述符集readfds、writefds、exceptfds中最大文件描述符值加1。

select的最后一个参数timeout代表最长阻塞时间。

select函数返回满足指定I/O要求的文件描述符的个数，如果是超时退出的，select函数返回0。如果出错，select函数返回-1。

有四个宏用来操作文件描述符集：

void FD_ZERO(fd_set *set); 清空一个文件描述符集。

void FD_SET(int fd, fd_set *set); 将一个文件描述符fd加入到指定的文件描述符集set中。

void FD_CLR(int fd, fd_set *set); 将一个文件描述符fd从指定的文件描述符集set中删除。

int FD_ISSET(int fd, fd_set *set); 测试文件描述符fd是否在指定的文件描述符集set中。

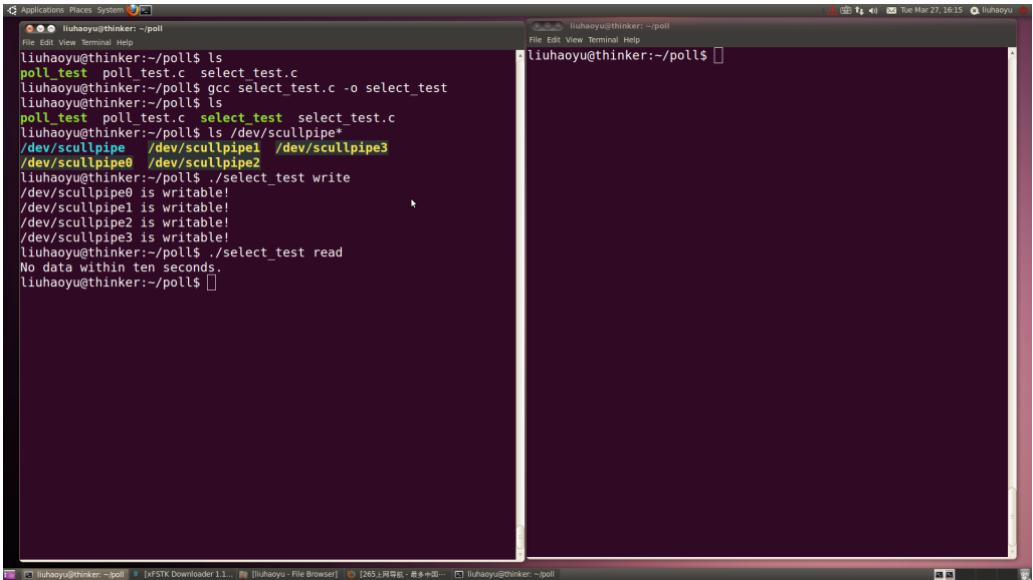
下面我们看使用select函数实现的测试驱动中poll操作的测试程序，其代码如下：

```
[cpp]
01. #include <stdio.h>
02. #include <stdlib.h>
03. #include <string.h>
04. #include <fcntl.h>
05. #include <unistd.h>
```

```
06. #include <sys/time.h>
07. #include <sys/types.h>
08. #include <sys/stat.h>
09.
10. int main(int argc, char *argv[])
11. {
12.     fd_set rfds, wfds;
13.     int fd0, fd1, fd2, fd3;
14.     char buf[100];
15.     int retval;
16.
17.     /* Wait up to ten seconds. */
18.     struct timeval tv;
19.     tv.tv_sec = 10;
20.     tv.tv_usec = 0;
21.
22.     if(argc != 2 || ((strcmp(argv[1], "read") != 0) && (strcmp(argv[1], "write") != 0)))
23.     {
24.         printf("usage: ./select_test read|write\n");
25.         return -1;
26.     }
27.
28.     fd0 = open("/dev/scullpipe0", O_RDWR);
29.     if ( fd0 < 0 )
30.     {
31.         printf("open scullpipe0 error\n");
32.         return -1;
33.     }
34.
35.     fd1 = open("/dev/scullpipe1", O_RDWR);
36.     if ( fd1 < 0 )
37.     {
38.         printf("open scullpipe1 error\n");
39.         return -1;
40.     }
41.
42.     fd2 = open("/dev/scullpipe2", O_RDWR);
43.     if ( fd2 < 0 )
44.     {
45.         printf("open scullpipe2 error\n");
46.         return -1;
47.     }
48.
49.     fd3 = open("/dev/scullpipe3", O_RDWR);
50.     if ( fd3 < 0 )
51.     {
52.         printf("open scullpipe3 error\n");
53.         return -1;
54.     }
55.
56.     if(strcmp(argv[1], "read") == 0)
57.     {
58.         FD_ZERO(&rfds);
59.         FD_SET(fd0, &rfds);
60.         FD_SET(fd1, &rfds);
61.         FD_SET(fd2, &rfds);
62.         FD_SET(fd3, &rfds);
63.         retval = select(fd3 + 1, &rfds, NULL, NULL, &tv);
64.     }
65.     else
66.     {
67.         FD_ZERO(&wfds);
68.         FD_SET(fd0, &wfds);
69.         FD_SET(fd1, &wfds);
70.         FD_SET(fd2, &wfds);
71.         FD_SET(fd3, &wfds);
72.         retval = select(fd3 + 1, NULL, &wfds, NULL, &tv);
73.     }
74.
75.     if (retval == -1)
76.     {
77.         printf("select error!\n");
78.         return -1;
79.     }
80.     else if (retval)
81.     {
82.         if(strcmp(argv[1], "read") == 0)
83.         {
84.             if(FD_ISSET(fd0, &rfds))
```

```
85.         {
86.             printf("/dev/scullpipe0 is readable!\n");
87.             memset(buf, 0, 100);
88.             read(fd0, buf, 100);
89.             printf("%s\n", buf);
90.         }
91.
92.         if(FD_ISSET(fd1, &rfd))
93.         {
94.             printf("/dev/scullpipe1 is readable!\n");
95.             memset(buf, 0, 100);
96.             read(fd1, buf, 100);
97.             printf("%s\n", buf);
98.         }
99.
100.        if(FD_ISSET(fd2, &rfd))
101.        {
102.            printf("/dev/scullpipe2 is readable!\n");
103.            memset(buf, 0, 100);
104.            read(fd2, buf, 100);
105.            printf("%s\n", buf);
106.        }
107.
108.        if(FD_ISSET(fd3, &rfd))
109.        {
110.            printf("/dev/scullpipe3 is readable!\n");
111.            memset(buf, 0, 100);
112.            read(fd3, buf, 100);
113.            printf("%s\n", buf);
114.        }
115.    }
116.    else
117.    {
118.        if(FD_ISSET(fd0, &wfd))
119.        {
120.            printf("/dev/scullpipe0 is writable!\n");
121.        }
122.
123.        if(FD_ISSET(fd1, &wfd))
124.        {
125.            printf("/dev/scullpipe1 is writable!\n");
126.        }
127.
128.        if(FD_ISSET(fd2, &wfd))
129.        {
130.            printf("/dev/scullpipe2 is writable!\n");
131.        }
132.
133.        if(FD_ISSET(fd3, &wfd))
134.        {
135.            printf("/dev/scullpipe3 is writable!\n");
136.        }
137.    }
138. }
139. else
140. {
141.     if(strcmp(argv[1], "read") == 0)
142.     {
143.         printf("No data within ten seconds.\n");
144.     }
145.     else
146.     {
147.         printf("Can not write within ten seconds.\n");
148.     }
149. }
150.
151. return 0;
152. }
```

测试过程如下图所示:



从上图可以看出，scullpipe0 - scullpipe3都是可写的。但是因为并没有向其中写入任何内容，所以读的时候会阻塞住，因为测试程序设置最长阻塞时间为10秒，所以10秒后解除阻塞退出，并打印”No data within ten seconds.”。

下面再次测试读操作，因为设备中没有内容，还是阻塞住，但是在10秒钟内，从另外一个终端向/dev/scullpipe2写入数据，这里是写入字符串”hello”，可以看到，写入数据后，测试程序解除阻塞，并打印”/dev/scullpipe2 is readable!”和”hello”字符串，这个过程如下图所示：

三、驱动程序中poll操作的实现

用户空间的poll和select函数，最终都会调用驱动程序中的poll函数，其函数原型如下：

```
[cpp]
01. unsigned int (*poll) (struct file *filp, poll_table *wait);
```

poll函数应该实现两个功能：

- 一是把能标志轮询状态变化的等待队列加入到poll_table中，这通过调用poll_wait函数实现。
- 二是返回指示能进行的I/O操作的标志位。

poll函数的第二个参数poll_table，是内核中用来实现poll，select系统调用的结构体，对于驱动开发者来说，不必关心其具体内容，可以把poll_table看成是不透明的结构体，只要拿过来使用就可以了。驱动程序通过poll_wait函数，把能够唤醒进程，改变轮询状态的等待队列加入到poll_table中。该函数定义如下：

[cpp]

```
01. void poll_wait (struct file *, wait_queue_head_t *, poll_table *);
```

对于poll函数的第二个功能，返回的标志位与用户空间相对应，最常用的标志位是POLLIN | POLLRDNORM和POLLOUT | POLLWRNORM，分别标志可进行非阻塞的读和写操作。

下面看scullpipe设备对poll操作的实现，其内容其实非常简单：

[cpp]

```
01. static unsigned int scull_p_poll(struct file *filp, poll_table *wait)
02. {
03.     struct scull_pipe *dev = filp->private_data;
04.     unsigned int mask = 0;
05.
06.     /*
07.      * The buffer is circular; it is considered full
08.      * if "wp" is right behind "rp" and empty if the
09.      * two are equal.
10.      */
11.     down(&dev->sem);
12.     poll_wait(filp, &dev->inq, wait);
13.     poll_wait(filp, &dev->outq, wait);
14.     if (dev->rp != dev->wp)
15.         mask |= POLLIN | POLLRDNORM; /* readable */
16.     if (spacefree(dev))
17.         mask |= POLLOUT | POLLWRNORM; /* writable */
18.     up(&dev->sem);
19.     return mask;
20. }
```

第239行，调用poll_wait函数将读等待队列加入到poll_table中。

第240行，调用poll_wait函数将写等待队列加入到poll_table中。

241 - 242行，如果有内容可读，设置可读标志位。

243 - 244行，如果有空间可写，设置可写标志位。

246行，将标志位返回。

驱动程序中的poll函数很简单，那么内核是怎么实现poll和select系统调用的呢？当用户空间程序调用poll和select函数时，内核会调用由用户程序指定的全部文件的poll方法，并向它们传递同一个poll_table结构。poll_table结构其实是一个生成“实际数据结构”的函数(名为poll_queue_proc)的封装，这个函数poll_queue_proc，不同的应用场景，内核有不同的实现，这里我们不仔细研究对应的函数。对于poll和select系统调用来说，这个“实际数据结构”是一个包含poll_table_entry结构的内存页链表。这里提到的相关数据结构在linux-2.6.32-38源码中，定义在include/linux/poll.h文件中，代码如下所示：

[cpp]

```
01. /*
02.  * structures and helpers for f_op->poll implementations
03.  */
04. #typedef void (*poll_queue_proc)
05.     (struct file *, wait_queue_head_t *, struct poll_table_struct *);
06. #typedef struct poll_table_struct {
07.     poll_queue_proc qproc;
08.     unsigned long key;
09. } poll_table;
10.
11. static inline void poll_wait(struct file * filp, wait_queue_head_t * wait_address, poll_table * pt)
12. {
13.     if (pt && wait_address)
14.         pt->qproc(filp, wait_address, pt);
15. }
16.
17. static inline void init_poll_funcptr(poll_table *pt, poll_queue_proc qproc)
18. {
19.     pt->qproc = qproc;
20.     pt->key = ~0UL; /* all events enabled */
21. }
22. 
```

```
23. 52struct poll_table_entry {
24. 53     struct file *filp;
25. 54     unsigned long key;
26. 55     wait_queue_t wait;
27. 56     wait_queue_head_t *wait_address;
28. 57};
```

poll操作我们就分析完了，内核要求驱动程序做的事并不多，但是我们在学习时，要把poll操作和前面介绍的阻塞型read/write函数以及scullpipe设备的等待队列等结合起来考虑，因为scullpipe设备是一个完整的程序模块。

更多 0

上一篇 LDD3源码分析之阻塞型I/O
下一篇 LDD3源码分析之异步通知

主题推荐 源码 数据结构 驱动开发 descriptor 数据

猜你在找

- linux poll 和 等待队列休眠的关系
- like,unlikely宏和GCC内建函数__builtin_expect()
- 线程之间的同步与互斥
- 通用GPIO模拟串口，提供源代码，本人经过测试OK。
- linux input输入子系统分析《二》：s3c2440的ADC简单
- 应届生去大公司与中小公司利弊之个人见解
- USB协议栈设备框架和连接枚举过程
- 如何在windows下面编译u-boot（原发于：2012-07-24
- intel dpdk api rte_eal_hugepage_init() 函数介绍
- u-boot编译笔记

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

2楼 雁子依然 2013-06-19 13:58发表
文字清晰，表述到位，受益匪浅，赞一个！

1楼 星期四 2012-08-07 17:59发表
good~

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

