

CSDN新首页上线啦，邀请你来立即体验！(http://blog.csdn.net/)

立即体



验

博客 (/blog.csdn.net?ref=toolbar)学院 (/edu.csdn.net?ref=toolbar)

下载 (/download.csdn.net?ref=toolbar)GitChat (/gitbook.cn/?ref=csdn)

更多

0



登录 (/passport.csdn.net/account/login?ref=toolbar)注册 (/passport.csdn.net/account/mobile/register?ref=toolbar&action=mobileRegister)

写 (/write.blog.csdn.net/?ref=toolbar) (/write.blog.csdn.net/?ref=toolbar)

在 Linux 下用户空间与内核空间数据交换的方式，第 2 部分: procfs、seq_file、debugfs和relayfs

转载

2011年05月15日 15:33:00

标签：file (/so.csdn.net/so/search/s.do?q=file&t=blog) /

linux (/so.csdn.net/so/search/s.do?q=linux&t=blog) /

struct (/so.csdn.net/so/search/s.do?q=struct&t=blog) /

null (/so.csdn.net/so/search/s.do?q=null&t=blog) / api (/so.csdn.net/so/search/s.do?q=api&t=blog) /

linux内核 (/so.csdn.net/so/search/s.do?q=linux内核&t=blog)

419

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！



banyao2006 (http://blog.c...

+ 关注

(http://blog.csdn.net/banyao2006)

码云

原创	粉丝	喜欢	未开通
16	40	0	(https://gite
			utm_sourc

他的最新文章

更多文章 (http://blog.csdn.net/banyao2006)

是的，我们SIP客户端打算采用resiproc
ate，媒体采用webrtc (http://blog.csdn.
net/banyao2006/article/details/517770
92)

freeswitch配置同个SIP账号注册多个终
端 (http://blog.csdn.net/banyao2006/art
icle/details/51775458)

pjsip,webrtc音视频解决方案 (http://blo
g.csdn.net/banyao2006/article/details/5



本系列文章包括两篇，它们详细地介绍了Linux系统下用户空间与内核空间数据交换的九种方式，包括内核启动参数、模块参数与sysfs、sysctl、系统调用、netlink、procfs、seq_file、debugfs和relayfs，并给出具体的例子帮助读者掌握这些技术的使用。

本文是该系列文章的第二篇，它介绍了procfs、seq_file、debugfs和relayfs，并结合给出的例子程序详细地说明了它们如何使用。

📌 procfs

procfs是比较老的一种用户态与内核态的数据交换方式，内核的很多数据都是通过这种方式出口给用户的，内核的很多参数也是通过这种方式来让用户方便设置的。除了sysctl出口到/proc下的参数，procfs提供的大部分内核参数是只读的。实际上，很多应用严重地依赖于procfs，因此它几乎是必不可少的组件。前面部分的几个例子实际上已经使用它来出口内核数据，但是并没有讲解如何使用，本节将讲解如何使用procfs。

Procfs提供了如下API：



```
struct proc_dir_entry *create_proc_entry(const char *name, mode_t mode,
                                         struct proc_dir_entry *parent)
```

该函数用于创建一个正常的proc条目，参数name给出要建立的proc条目的名称，参数mode给出了建立的该proc条目的访问权限，参数parent指定建立的proc条目所在的目录。如果要在/proc下建立proc条目，parent应当为NULL。否则它应当为proc_mkdir返回的struct proc_dir_entry结构的指针。

```
extern void remove_proc_entry(const char *name, struct proc_dir_entry *parent)
```

该函数用于删除上面函数创建的proc条目，参数name给出要删除的proc条目的名称，参数parent指定建立的proc条目所在的目录。

```
struct proc_dir_entry *proc_mkdir(const char * name, struct proc_dir_entry *parent)
```

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

1114392)

(<https://passport.csdn.net/a>

内存管理 (<http://blog.csdn.net/banyao2006/article/details/49700001>)

阐述信号与系统中三大变换（即傅里叶变换、拉普拉斯变换、Z变换）的关系 (<http://blog.csdn.net/banyao2006/article/details/49680925>)

相关推荐

【转载】<http://blog.csdn.net/banyao2006/article/details/49700001> 在 Linux 下用户空间与内核空间数据交换的方式/第 2 部分: procfs、seq

在 Linux 下用户空间与内核空间数据交换的方式/第 2 部分: procfs、seq_file、de

在 Linux 下用户空间与内核空间数据交换的方式/第 2 部分: procfs、seq_file、de

在 Linux 下用户空间与内核空间数据交换的方式/第 2 部分: procfs、seq_file、de

该函数用于创建一个proc目录，参数name指定要创建的proc目录的名称，参数parent为该proc目录所在的目录。

```
extern struct proc_dir_entry *proc_mkdir_mode(const char *name, mode_t mode,
                                              struct proc_dir_entry *parent);
struct proc_dir_entry *proc_symlink(const char * name,
                                     struct proc_dir_entry * parent, const char * dest)
```



该函数用于建立一个proc条目的符号链接，参数name给出要建立的符号链接proc条目的名称，参数parent指定符号连接所在的目录，参数dest指定链接到的proc条目名称。



```
struct proc_dir_entry *create_proc_read_entry(const char *name,
                                              mode_t mode, struct proc_dir_entry *base,
                                              read_proc_t *read_proc, void * data)
```

该函数用于建立一个规则的只读proc条目，参数name给出要建立的proc条目的名称，参数mode给出了建立的该proc条目的访问权限，参数base指定建立的proc条目所在的目录，参数read_proc给出读去该proc条目的操作函数，参数data为该proc条目的专用数据，它将保存在该proc条目对应的struct file结构的private_data字段中。

```
struct proc_dir_entry *create_proc_info_entry(const char *name,
                                              mode_t mode, struct proc_dir_entry *base, get_info_t *get_info)
```

该函数用于创建一个info型的proc条目，参数name给出要建立的proc条目的名称，参数mode给出了建立的该proc条目的访问权限，参数base指定建立的proc条目所在的目录，参数get_info指定该proc条目的get_info操作函数。实际上get_info等同于read_proc，如果proc条目没有定义个read_proc，对该proc条目的read操作将使用get_info取代，因此它在功能上非常类似于函数create_proc_read_entry。

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！



(https://passport.csdn.net/a

达内证书有用吗



奥迪r8二手



loft公寓出租

酒店式公寓月租 魔方公寓 一点点加..

酒店式公寓出租 免费云主机试用一年

单身公寓 一室一厅出租 38平小户型..

移民澳大利亚 语音识别 70平小户型..

他的热门文章

qemu (http://blog.csdn.net/banyao2006/article/details/7352779)

7111

在 Android上實作一個FFmpeg+SDL的media player (http://blog.csdn.net/banyao2006/article/details/6126252)

5088

登录

注册



```
struct proc_dir_entry *proc_net_create(const char *name,
                                       mode_t mode, get_info_t *get_info)
```

该函数用于在/proc/net目录下创建一个proc条目，参数name给出要建立的proc条目的名称，参数mode给出了建立的该proc条目的访问权限，参数get_info指定该proc条目的get_info操作函数。

```
struct proc_dir_entry *proc_net_fops_create(const char *name,
                                             mode_t mode, struct file_operations *fops)
```

该函数也用于在/proc/net下创建proc条目，但是它也同时指定了对该proc条目的文件操作函数。

```
void proc_net_remove(const char *name)
```

该函数用于删除前面两个函数在/proc/net目录下创建的proc条目。参数name指定要删除的proc名称。除了这些函数，值得一提的是结构struct proc_dir_entry，为了创建一了可写的proc条目并指定该proc条目的写操作函数，必须设置上面的这些创建proc条目的函数返回的指针指向的struct proc_dir_entry结构的write_proc字段，并指定该proc条目的访问权限有写权限。

为了使用这些接口函数以及结构struct proc_dir_entry，用户必须在模块中包含头文件linux/proc_fs.h。在源代码包中给出了procfs示例程序procfs_exam.c，它定义了三个proc文件条目和一个proc目录条目，读者在插入该模块后应当看到如下结构：

```
$ ls /proc/myproctest
aint          astring          bigprocfile
$
```

读者可以通过cat和echo等文件操作函数来查看和设置这些proc文件。特别需要指出，bigprocfile是一个大文件（超过一个内存页），对于这种大文件，procfs有一些限制，因为它提供的缓存，只有一个页，因此必须特别小心，并对超过页的部分做特别的考虑，处理起来比较复杂并且很容易出错，所有procfs并不适合于大数据量输入输出。后面推荐seq_file就是因为这个缺陷而设计的，当然seq_file依赖于procfs的一些基础功能。

將 SDL 整合至Android平台(<https://passport.csdn.net/article/details/6126247>)

4394

將 FFmpeg 整合至Android平台 (<http://blog.csdn.net/banyao2006/article/details/6126244>)

3601

Lucas-Kanade算法原理介绍及OpenCV代码实现分析 (<http://blog.csdn.net/banyao2006/article/details/39484113>)

3434

加入CSDN，享受更精准的内容推荐，与5000万程序猿共同成长！

登录

注册



✚ 回页首 (<http://www.ibm.com/developerworks/cn/linux/l-kerns-usrs2/#main>)

二、seq_file

一般地，内核通过在procfs文件系统下建立文件来向用户空间提供输出信息，用户空间可以通过任何文本阅读应用查看该文件信息，但是procfs有一个缺陷，如果输出内容大于1个内存页，需要多次读，因此处理起来很难，另外，如果输出太大，速度比较慢，有时会出现一些意想不到的情况，Alexander Viro实现了一套新的功能，使得内核输出大文件信息更容易，该功能出现在2.4.15（包括2.4.15）以后的所有2.4内核以及2.6内核中，尤其是在2.6内核中，已经大量地使用了该功能。

要想使用seq_file功能，开发者需要包含头文件linux/seq_file.h，并定义与设置一个seq_operations结构（类似于file_operations结构）：



```
struct seq_operations {
    void * (*start) (struct seq_file *m, loff_t *pos);
    void (*stop) (struct seq_file *m, void *v);
    void * (*next) (struct seq_file *m, void *v, loff_t *pos);
    int (*show) (struct seq_file *m, void *v);
};
```

start函数用于指定seq_file文件的读开始位置，返回实际读开始位置，如果指定的位置超过文件末尾，应当返回NULL，start函数可以有一个特殊的返回SEQ_START_TOKEN，它用于让show函数输出文件头，但这只能在pos为0时使用，next函数用于把seq_file文件的当前读位置移动到下一个读位置，返回实际的下一个读位置，如果已经到达文件末尾，返回NULL，stop函数用于在读完seq_file文件后调用，它类似于文件操作close，用于做一些必要的清理，如释放内存等，show函数用于格式化输出，如果成功返回0，否则返回出错码。

Seq_file也定义了一些辅助函数用于格式化输出：

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

登录

注册



函数seq_putc用于把一个字符输出到seq_file文件。

```
int seq_puts(struct seq_file *m, const char *s);
```

函数seq_puts则用于把一个字符串输出到seq_file文件。



0

```
int seq_escape(struct seq_file *, const char *, const char *);
```



函数seq_escape类似于seq_puts，只是，它将把第一个字符串参数中出现的包含在第二个字符串参数中的字符按照八进制形式输出，也即对这些字符进行转义处理。



```
int seq_printf(struct seq_file *, const char *, ...)
    __attribute__((format (printf,2,3)));
```

函数seq_printf是最常用的输出函数，它用于把给定参数按照给定的格式输出到seq_file文件。

```
int seq_path(struct seq_file *, struct vfsmount *, struct dentry *, char *);
```

函数seq_path则用于输出文件名，字符串参数提供需要转义的文件名字符，它主要供文件系统使用。

在定义了结构struct seq_operations之后，用户还需要把打开seq_file文件的open函数，以便该结构与对应于seq_file文件的struct file结构关联起来，例如，struct seq_operations定义为：

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

登录

注册



(https://passport.csdn.net/a

```
struct seq_operations exam_seq_ops = {
    .start = exam_seq_start,
    .stop = exam_seq_stop,
    .next = exam_seq_next,
    .show = exam_seq_show
};
```

那么，open函数应该如下定义：



```
0
static int exam_seq_open(struct inode *inode, struct file *file)
{
    return seq_open(file, &exam_seq_ops);
};
```



注意，函数seq_open是seq_file提供的函数，它用于把struct seq_operations结构与seq_file文件关联起来。

最后，用户需要如下设置struct file_operations结构：

```
struct file_operations exam_seq_file_ops = {
    .owner    = THIS_MODULE,
    .open     = exm_seq_open,
    .read     = seq_read,
    .llseek   = seq_lseek,
    .release  = seq_release
};
```

注意，用户仅需要设置open函数，其它的都是seq_file提供的函数。

然后，用户创建一个/proc文件并把它文件操作设置为exam_seq_file_ops即可：

```
struct proc_dir_entry *entry;
entry = create_proc_entry("exam_seq_file", 0, NULL);
if (entry)
```

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

[登录](#)
[注册](#)


对于简单的输出，seq_file用户并不需要定义和设置这么多函数与结构，它仅需定义一个show函数，然后使用single_open来定义open函数就可以，以下是使用这种简单形式的一般步骤：

1. 定义一个show函数

```
int exam_show(struct seq_file *p, void *v)
{
    // ...
    return 0;
}
```

2. 定义open函数

```
int exam_single_open(struct inode *inode, struct file *file)
{
    return(single_open(file, exam_show, NULL));
}
```

注意要使用single_open而不是seq_open。

3. 定义struct file_operations结构

```
struct file_operations exam_single_seq_file_operations = {
    .open      = exam_single_open,
    .read      = seq_read,
    .llseek    = seq_lseek,
    .release   = single_release,
};
```

注意，如果open函数使用了single_open，release函数必须为single_release，而不是seq_release。在源代码包中给出了一个使用seq_file的具体例子seqfile_exam.c，它使用seq_file提供了一个查看当前系统运行的所有进程的/proc接口，在编译并插入该模块后，用户通过命令"cat /proc/ exam_esq_file"可以查看系统的所有进程。

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

[登录](#)
[注册](#)


✦ 回首页 (<http://www.ibm.com/developerworks/cn/linux/l-kerns-usrs2/#main>)

三、debugfs

内核开发者经常需要向用户空间应用输出一些调试信息，在稳定的系统中可能根本不需要这些调试信息，但是在开发过程中，为了搞清楚内核的行为，调试信息非常必要，printk可能是用的最多的，但它并不是最好的，调试信息只是在开发中用于调试，而printk将一直输出，因此开发完毕后需要清除不必要的printk语句，另外如果开发者希望用户空间应用能够改变内核行为时，printk就无法实现。因此，需要一种新的机制，那只有在需要的时候使用，它在需要时通过在一个虚拟文件系统中创建一个或多个文件来向用户空间应用提供调试信息。

有几种方式可以实现上述要求：

使用procfs，在/proc创建文件输出调试信息，但是procfs对于大于一个内存页（对于x86是4K）的输出比较麻烦，而且速度慢，有时会出现一些意想不到的问题。

使用sysfs（2.6内核引入的新的虚拟文件系统），在很多情况下，调试信息可以存放在那里，但是sysfs主要用于系统管理，它希望每一个文件对应内核的一个变量，如果使用它输出复杂的数据结构或调试信息是非常困难的。

使用libfs创建一个新的文件系统，该方法极其灵活，开发者可以为新文件系统设置一些规则，使用libfs使得创建新文件系统更加简单，但是仍然超出了一个开发者的想象。

为了使得开发者更加容易使用这样的机制，Greg Kroah-Hartman开发了debugfs（在2.6.11中第一次引入），它是一个虚拟文件系统，专门用于输出调试信息，该文件系统非常小，很容易使用，可以在配置内核时选择是否构件到内核中，在不选择它的情况下，使用它提供的API的内核部分不需要做任何改动。

使用debugfs的开发者首先需要在文件系统中创建一个目录，下面函数用于在debugfs文件系统下创建一个目录：

```
struct dentry *debugfs_create_dir(const char *name, struct dentry *parent);
```

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

登录

注册



(https://passport.csdn.net/a

参数name是要创建的目录名，参数parent指定创建目录的父目录的dentry，如果为NULL，目录将创建在debugfs文件系统的根目录下。如果返回为-ENODEV，表示内核没有把debugfs编译到其中，如果返回为NULL，表示其他类型的创建失败，如果创建目录成功，返回指向该目录对应的dentry条目的指针。

下面函数用于在debugfs文件系统中创建一个文件：

```
struct dentry *debugfs_create_file(const char *name, mode_t mode,
                                   struct dentry *parent, void *data,
                                   struct file_operations *fops);
```

参数name指定要创建的文件名，参数mode指定该文件的访问许可，参数parent指向该文件所在目录，参数data为该文件特定的一些数据，参数fops为实现在该文件上进行文件操作的file_operations结构指针，在很多情况下，由seq_file（前面章节已经讲过）提供的文件操作实现就足够了，因此使用debugfs很容易，当然，在一些情况下，开发者可能仅需要使用用户应用可以控制的变量来调试，debugfs也提供了4个这样的API方便开发者使用：

```
struct dentry *debugfs_create_u8(const char *name, mode_t mode,
                                 struct dentry *parent, u8 *value);
struct dentry *debugfs_create_u16(const char *name, mode_t mode,
                                  struct dentry *parent, u16 *value);
struct dentry *debugfs_create_u32(const char *name, mode_t mode,
                                  struct dentry *parent, u32 *value);
struct dentry *debugfs_create_bool(const char *name, mode_t mode,
                                   struct dentry *parent, u32 *value);
```

参数name和mode指定文件名和访问许可，参数value为需要让用户应用控制的内核变量指针。

当内核模块卸载时，Debugfs并不会自动清除该模块创建的目录或文件，因此对于创建的每一个文件或目录，开发者必须调用下面函数清除：

```
void debugfs_remove(struct dentry *dentry);
```

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

[登录](#)
[注册](#)


参数dentry为上面创建文件和目录的函数返回的dentry指针。

在源代码包中给出了一个使用debugfs的示例模块debugfs_exam.c，为了保证该模块正确运行，必须让内核支持debugfs，debugfs是一个调试功能，因此它位于主菜单Kernel hacking，并且必须选择Kernel debugging选项才能选择，它的选项名称为Debug Filesystem。为了在用户态使用debugfs，用户必须mount它，下面是在作者系统上的使用输出：



```
$0mkdir -p /debugfs
$ mount -t debugfs debugfs /debugfs
$ insmod ./debugfs_exam.ko
$ ls /debugfs
debugfs-exam
$ ls /debugfs/debugfs-exam
u8_var          u16_var          u32_var          bool_var
$ cd /debugfs/debugfs-exam
$ cat u8_var
0
$ echo 200 > u8_var
$ cat u8_var
200
$ cat bool_var
N
$ echo 1 > bool_var
$ cat bool_var
Y
```

➤ 回首页 (<http://www.ibm.com/developerworks/cn/linux/l-kerns-usrs2/#main>)

四、relayfs

relayfs是一个快速的转发（relay）数据的文件系统，它以其功能而得名。它为那些需要从内核空间转发大量数据到用户空间精确的真相推荐提供了快速有效的转发机制！

登录

注册



(https://passport.csdn.net/a

Channel是relayfs文件系统定义的一个主要概念，每一个channel由一组内核缓存组成，每一个CPU有一个对应于该channel的内核缓存，每一个内核缓存用一个在relayfs文件系统中的文件文件表示，内核使用relayfs提供的写函数把需要转发给用户空间的数据快速地写入当前CPU上的channel内核缓存，用户空间应用通过标准的文件I/O函数在对应的channel文件中可以快速地取得这些被转发出的数据mmap来。写入到channel中的数据格式完全取决于内核中创建channel的模块或子系统。

relayfs的用户空间API：

relayfs实现了四个标准的文件I/O函数，open、mmap、poll和close

- open()，打开一个channel在某一个CPU上的缓存对应的文件。
- mmap()，把打开的channel缓存映射到调用者进程的内存空间。
- read()，读取channel缓存，随后的读操作将看不到被该函数消耗的字节，如果channel的操作模式为非覆盖写，那么用户空间应用在有内核模块写时仍可以读取，但是如果channel的操作模式为覆盖式，那么在读操作期间如果有内核模块进行写，结果将无法预知，因此对于覆盖式写的channel，用户应当在确认在channel的写完全结束后再进行读。
- poll()，用于通知用户空间应用转发数据跨越了子缓存的边界，支持的轮询标志有POLLIN、POLLRDNORM和POLLERR。
- close()，关闭open函数返回的文件描述符，如果没有进程或内核模块打开该channel缓存，close函数将释放该channel缓存。

注意：用户态应用在使用上述API时必须保证已经挂载了relayfs文件系统，但内核在创建和使用channel时不需要relayfs已经挂载。下面命令将把relayfs文件系统挂载到/mnt/relay。

```
mount -t relayfs relayfs /mnt/relay
```

relayfs内核API：

relayfs提供给内核的API包括四类：channel管理、写函数、回调函数和辅助函数。

Channel管理函数包括：

- relay_open(base_filename, parent, subbuf_size, n_subbufs, overwrite, callbacks)
- relay_close(chan)
- relay_flush(chan)

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

登录

注册



(https://passport.csdn.net/a

- relayfs_create_dir(name, parent)
- relayfs_remove_dir(dentry)
- relay_commit(buf, reserved, count)
- relay_subbufs_consumed(chan, cpu, subbufs_consumed)

写函数包括：

- relay_write(chan, data, length)
- __relay_write(chan, data, length)
- relay_reserve(chan, length)

回调函数包括：

- subbuf_start(buf, subbuf, prev_subbuf_idx, prev_subbuf)
- buf_mapped(buf, filp)
- buf_unmapped(buf, filp)

辅助函数包括：

- relay_buf_full(buf)
- subbuf_start_reserve(buf, length)

前面已经讲过，每一个channel由一组channel缓存组成，每个CPU对应一个该channel的缓存，每一个缓存又由一个或多个子缓存组成，每一个缓存是子缓存组成的一个环型缓存。

函数relay_open用于创建一个channel并分配对应于每一个CPU的缓存，用户空间应用通过在relayfs文件系统中对应的文件可以访问channel缓存，参数base_filename用于指定channel的文件名，relay_open函数将在relayfs文件系统中创建base_filename0..base_filenameN-1，即每一个CPU对应一个channel文件，其中N为CPU数，缺省情况下，这些文件将建立在relayfs文件系统的根目录下，但如果参数parent非空，该函数将把channel文件创建于parent目录下，parent目录使用函数relay_create_dir创建，函数relay_remove_dir用于删除由函数relay_create_dir创建的目录，谁创建的目录，谁就负责在不用时负责删除。参数subbuf_size用于指定channel缓存中每一个子缓存的大小，参数n_subbufs用于指定channel缓存包含的子缓存数，因此实际的channel缓存大小为(subbuf_size x n_subbufs)，参数overwrite用于指定该channel的操作模式，relayfs提供了两种写模式，一种是覆盖式写，另一种是非覆盖式写。使用哪一种模式完全取决于函数subbuf_start的实现，覆盖写将在缓存已满的情况下无条件地继续从缓存的开始写数据，而不管这些数据是否已经被用户应用读取，因此写操作决不失败。在非覆盖写模式下，如果缓存满了，写将失败，但内核将在用户空间应用读取缓存数据时通过函数relay_subbufs_consumed()通知relayfs。如果用户空间应用没来得及消耗缓存中的数据或缓存已满，两种模式都将导致数据丢失，唯一的区别是，前者丢失数据在缓存开头，而后者丢失数据在缓存末尾。一旦内核再次调用函数relay_subbufs_consumed()，已满的缓存将

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

登录

注册



<https://passport.csdn.net/a>

不再满，因而可以继续写该缓存。当缓存满了以后，relayfs将调用回调函数buf_full()来通知内核模块或子系统。当新的数据太大无法写入当前子缓存剩余的空间时，relayfs将调用回调函数subbuf_start()来通知内核模块或子系统将需要使用新的子缓存。内核模块需要在该回调函数中实现下述功能：

初始化新的子缓存；

如果1正确，完成当前子缓存；

如果2正确，返回是否正确完成子缓存切换；

在非覆盖写模式下，回调函数subbuf_start()应该如下实现：



0

```
static int subbuf_start(struct rchan_buf *buf,
                        void *subbuf,
                        void *prev_subbuf,
                        unsigned int prev_padding)
{
    if (prev_subbuf)
        *((unsigned *)prev_subbuf) = prev_padding;
    if (relay_buf_full(buf))
        return 0;
    subbuf_start_reserve(buf, sizeof(unsigned int));
    return 1;
}
```

如果当前缓存满，即所有的子缓存都没读取，该函数返回0，指示子缓存切换没有成功。当子缓存通过函数relay_subbufs_consumed()被读取后，读取者将负责通知relayfs，函数relay_buf_full()在已经有读者读取子缓存数据后返回0，在这种情况下，子缓存切换成功进行。

在覆盖写模式下，subbuf_start()的实现与非覆盖模式类似：

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

登录

注册



(https://passport.csdn.net/a

```
static int subbuf_start(struct rchan_buf *buf,
                        void *subbuf,
                        void *prev_subbuf,
                        unsigned int prev_padding)
{
    if (prev_subbuf)
        *((unsigned *)prev_subbuf) = prev_padding;
    subbuf_start_reserve(buf, sizeof(unsigned int));
    return 1;
}
```



0

只是不做relay_buf_full()检查，因为此模式下，缓存是环行的，可以无条件地写。因此在此模式下，子缓存切换必定成功，函数relay_subbufs_consumed()也无须调用。如果channel写者没有定义subbuf_start()，缺省的实现将被使用。可以通过在回调函数subbuf_start()中调用辅助函数subbuf_start_reserve()在子缓存中预留头空间，预留空间可以保存任何需要的信息，如上面例子中，预留空间用于保存子缓存填充字节数，在subbuf_start()实现中，前一个子缓存的填充值被设置。前一个子缓存的填充值和指向前一个子缓存的指针一道作为subbuf_start()的参数传递给subbuf_start()，只有在子缓存完成后，才能知道填充值。subbuf_start()也被在channel创建时分配每一个channel缓存的第一个子缓存时调用，以便预留头空间，但在这种情况下，前一个子缓存指针为NULL。

内核模块使用函数relay_write()或__relay_write()往channel缓存中写需要转发的数据，它们的区别是前者失效了本地中断，而后者只抢占失效，因此前者可以在任何内核上下文安全使用，而后者应当在没有任何中断上下文将写channel缓存的情况下使用。这两个函数没有返回值，因此用户不能直接确定写操作是否失败，在缓存满且写模式为非覆盖模式时，relayfs将通过回调函数buf_full来通知内核模块。

函数relay_reserve()用于在channel缓存中预留一段空间以便以后写入，在那些没有临时缓存而直接写入channel缓存的内核模块可能需要该函数，使用该函数的内核模块在实际写这段预留的空间时可以通过调用relay_commit()来通知relayfs。当所有预留的空间全部写完并通过relay_commit通知relayfs后，relayfs将调用回调函数deliver()通知内核模块一个完整的子缓存已经填满。由于预留空间的操作并不在写channel的内核模块完全控制之下，因此relay_reserve()不能很好地保护缓存，因此当内核模块调用relay_reserve()时必须采取恰当的同步机制。

当内核模块结束对channel的使用后需要调用relay_close()来关闭channel，如果没有任何用户在引用该channel，它将和对应的缓存全部被释放。

函数relay_flush()强制在所有的channel缓存上做一个子缓存切换，它在channel被关闭前使用来终止和处理加入最后的子缓存。

登录

注册



(https://passport.csdn.net/a

函数relay_reset()用于将一个channel恢复到初始状态，因而不必释放现存的内存映射并重新分配新的channel缓存就可以使用channel，但是该调用只有在该channel没有任何用户在写的情况下才可以安全使用。

回调函数buf_mapped()在channel缓存被映射到用户空间时被调用。

回调函数buf_unmapped()在释放该映射时被调用。内核模块可以通过它们触发一些内核操作，如开始或结束channel写操作。

在源代码包中给出了一个使用relayfs的示例程序relayfs_exam.c，它只包含一个内核模块，对于复杂的使用，需要应用程序配合。该模块实现了类似于文章中seq_file示例实现的功能。

当然为了使用relayfs，用户必须让内核支持relayfs，并且要mount它，下面是作者系统上的使用该模块的输出信息：

```
$ mkdir -p /relayfs
$ insmod ./relayfs-exam.ko
$ mount -t relayfs relayfs /relayfs
$ cat /relayfs/example0
...
$
```

relayfs是一种比较复杂的内核态与用户态的数据交换方式，本例子程序只提供了一个较简单的使用方式，对于复杂的使用，请参考relayfs用例页面<http://relayfs.sourceforge.net/examples.html> (<http://relayfs.sourceforge.net/examples.html>)。

[+ 回首页 \(http://www.ibm.com/developerworks/cn/linux/l-kerns-usrs2/#main\)](http://www.ibm.com/developerworks/cn/linux/l-kerns-usrs2/#main)

小结

本文是该系列文章最后一篇，它详细地讲解了其余四种用户空间与内核空间的数据交换方式，并通过实际例子程序向读者讲解了如何在内核开发中使用这些技术，其中seq_file是单向的，即只能向内核传递，而不能从内核获取，而另外三种方式均可以进行双向数据交换，即既可以从用户应用传递给内核，又可以从内核传递给用户应用。

加入CSDN给应用更精准的内容推荐，由5000万程序员共同组成！

登录

注册



核行为。seq_file实际上依赖于procfs，因此为了使用seq_file，必须使内核支持procfs。debugfs用于内核开发者调试使用，它比其他集中方式都方便，但是仅用于简单类型的变量处理。relayfs是一种非常复杂的数据交换方式，要想准确使用并不容易，但是如果使用得当，它远比procfs和seq_file功能强大。

(<https://passport.csdn.net/a>

参考资料

1. Linux 2.6.13的内核源码，<http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.13.tar.bz2>
 (<http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.13.tar.bz2>)。
- ⁰ 2. Linux 2.6.14的内核源码，<http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.14.tar.bz2>
 (<http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.14.tar.bz2>)。
3. 内核文档，Documentation/filesystems/proc.txt。
-  4. LWN, Driver porting: The seq_file interface, <http://lwn.net/Articles/22355/>
(<http://lwn.net/Articles/22355/>)。
5. LWN, debugfs, <http://lwn.net/Articles/115405/> (<http://lwn.net/Articles/115405/>)。
-  6. Linux kernel procfs guide, <http://kernelnewbies.org/documents/kdoc/procfs-guide/lkprocfsguide.html>
(<http://kernelnewbies.org/documents/kdoc/procfs-guide/lkprocfsguide.html>)。
7. 内核文档，Documentation/filesystems/relayfs.txt。
8. Relayfs project homepage, <http://relayfs.sourceforge.net/> (<http://relayfs.sourceforge.net/>)。
9. relayfs usage examples, <http://relayfs.sourceforge.net/examples.html>
(<http://relayfs.sourceforge.net/examples.html>)。
10. relayfs source, <http://prdownloads.sourceforge.net/relayfs/old-relayfs-051205-kernel-2.6.11.patch.bz2?download> (<http://prdownloads.sourceforge.net/relayfs/old-relayfs-051205-kernel-2.6.11.patch.bz2?download>)。
11. relayfs example applications, <http://prdownloads.sourceforge.net/relayfs/relay-apps-0.91.tar.gz?download> (<http://prdownloads.sourceforge.net/relayfs/relay-apps-0.91.tar.gz?download>)。

关于作者

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

登录

注册



杨燚，计算机科学硕士，毕业于中科院计算技术研究所，有4年的Linux内核编程经验，目前从事嵌入式实时Linux的开发与性能测试。您可以通过 yang.y.yi@gmail.com (<mailto:yang.y.yi@gmail.com?cc=>) 与作者联系。

(<https://passport.csdn.net/a>



0



相关文章推荐

【转载】在 Linux 下用户空间与内核空间数据交换的方式，第 2 部分: procfs、seq_file、deb...

燚 杨 (yang.y.yi@gmail.com), 计算机科学硕士 简介：本系列文章包括两篇，它们文详细地地介绍了Linux系统下用户空间与内核空间数据交换的九种方式，包括内核启动参数、模块参...



[cuijianzhongswust](http://blog.csdn.net/cuijianzhongswust) (<http://blog.csdn.net/cuijianzhongswust>) 2011年10月18日 19:52 1200

在 Linux 下用户空间与内核空间数据交换的方式，第 2 部分: procfs、seq_file、debugfs和rel...

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！


登录

注册



<http://www.ibm.com/developerworks/cn/linux/l-kerns-usrs2/index.html> 简介：本系列文章包括两篇，它们详细地介绍了Linux...

(<https://passport.csdn.net/a>

 beckdon (<http://blog.csdn.net/beckdon>) 2013年09月02日 19:43  397



【前端逆袭记】我是怎么从月薪4k到40k的！



谨以此篇文章献给我奋斗过的程序人生！我第一次编码是在我大一的时候....

(http://www.baidu.com/cb.php?c=lgF_pyfqHmknj0dP1f0IZ0qnfK9ujYzP1ndPWb10Aw-5Hc3rHnYnHb0TAq15HfLPWRznjb0T1YdryR1nW9BmH64PjT1n16v0AwY5HDdnHn3rjcknWT0IgF_5y9YIZ0IQzq-uZR8uLpUB48ugfEIAqspynElvNBnHqDIAdxTvqdThP-5yF_UvTkn0KzujYk0AFV5H00TZcqn0KdpyfqHRLPjnvnfKEpyfqHc4rj6kP0KWpyfqP1cvrHnz0AqLUWYs0ZK45HcsP6KWThnqPWn1n10)




在 Linux 下用户空间与内核空间数据交换的方式，第 2 部分: procfs、seq_file、debugfs和rel...

转载地址:<http://www.ibm.com/developerworks/cn/linux/l-kerns-usrs2/index.html> 蔡 杨 (yang.y.yi@gma...

 liangxiaozhang (<http://blog.csdn.net/liangxiaozhang>) 2013年01月06日 16:44  496

在 Linux 下用户空间与内核空间数据交换的方式，第 2 部分: procfs、seq_file、debugfs和rel...

一、procfs procfs是比较老的一种用户态与内核态的数据交换方式，内核的很多数据都是通过这种方式出口给用户的，内核的很多参数也是通过这种方式来让用户方便设置的。除了sysctl出口到/p...

 yangmingjinqq (<http://blog.csdn.net/yangmingjinqq>) 2016年05月12日 10:16  231

用户与内核空间数据交换的方式(3)-seq_file (<http://blog.csdn.net/kudingcha5279/article/details/20130417-2551.html> 一般地，内核通过在pro...

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！
原文：<http://www.beedon.cn/html/yangmingjinqq/2016/17-2551.html>

登录

注册





 kudingcha5279 (<http://blog.csdn.net/kudingcha5279>) 2013年08月06日 00:25  358

(<https://passport.csdn.net/a>

<p>7 </p> <p>122.00/套 供应 埋地 防水接线盒 HKA-3直通型 埋地灌</p>	<p>8 </p> <p>0.03/个 供应光纤热缩套管/单 芯热缩管/60mm热缩</p>	<p>9 </p> <p>1.00/米 高价上门回收光缆4 6 8 12 24 48 72 96 144</p>
--	--	--



用户空间与内核空间数据交换的方式(2)-----procfs (<http://blog.csdn.net/liangxiaozhang/artic...>)

转载地址:<http://www.cnblogs.com/hoys/archive/2011/04/10/2011141.html> procfs是比较老的一种用户态与内核态的数据交换方式，内核...

 liangxiaozhang (<http://blog.csdn.net/liangxiaozhang>) 2013年01月06日 16:16  361



用户空间与内核空间数据交换的方式-----seq_file (http://blog.csdn.net/wh8_2011/article/det...)

一般地，内核通过在procfs文件系统下建立文件来向用户空间提供输出信息，用户空间可以通过任何文本阅读应用查看该文件信息，但是procfs 有一个缺陷，如果输出内容大于1个内存页，需要多次读，因此处理...

 wh8_2011 (http://blog.csdn.net/wh8_2011) 2016年01月18日 09:54  241

用户空间与内核空间数据交换的方式(2)-----procfs (http://blog.csdn.net/Tommy_wxie/article...)

procfs是比较老的一种用户态与内核态的数据交换方式，内核的很多数据都是通过 这种方式出口给用户的，内核的很多参数也是通过这种方式来让用户方便设置的。除了sysctl出口到/proc下的参数，pro...

 Tommy_wxie (http://blog.csdn.net/Tommy_wxie) 2012年10月04日 14:12  1012

用户空间与内核空间数据交换的方式(2)-----procfs (http://blog.csdn.net/liuqiang_mail/article...)

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

登录

注册




procfs是比较老的一种用户态与内核态的数据交换方式，内核的很多数据都是通过这种方式出口给用户的，内核的很多参数也是通过这种方式来让用户方便设置的。除了sysctl出口到/proc下的参数，proc...

(https://passport.csdn.net/a

 liuqiang_mail (http://blog.csdn.net/liuqiang_mail) 2012年08月21日 13:58 313

用户空间与内核空间数据交换的方式(2)-----procfs (<http://blog.csdn.net/pillarbuaa/article/de...>


 <http://yuxu9710108.blog.163.com/blog/static/23751534201110984955374/> 用户空间与内核空间数据交换的方式(2)-----pro...

 pillarbuaa (<http://blog.csdn.net/pillarbuaa>) 2012年07月16日 09:53 770




用户与内核空间数据交换的方式(4)-relayfs (<http://blog.csdn.net/kudingcha5279/article/deta...>

原文：<http://www.embeddedlinux.org.cn/html/yingjianqudong/201304/17-2550.html> relayfs是一个快速的转发（re...

 kudingcha5279 (<http://blog.csdn.net/kudingcha5279>) 2013年08月06日 00:26 446

用户空间与内核空间数据交换的方式(3)-----seq_file (<http://blog.csdn.net/liangxiaozhang/art...>

转载地址:<http://www.cnblogs.com/hoys/archive/2011/04/10/2011261.html> 一般地，内核通过在procfs文件系统下建立文件来向用户空间...

 liangxiaozhang (<http://blog.csdn.net/liangxiaozhang>) 2013年01月06日 16:17 326

用户空间与内核空间数据交换的方式(3)-----seq_file (<http://blog.csdn.net/chinseeker/article/...>

原文：<http://hi.baidu.com/zhaoercheng/blog/item/aff223f4ad796f3d730eec44.html> 一般地，内核通过在procfs文件系...

 chinseeker (<http://blog.csdn.net/chinseeker>) 2012年07月19日 21:31 392

加入CSDN，享受更精准的内容推荐，与5000万程序员共同成长！

登录

注册

