# Seq\_file

本文主要翻译linux-asic/Documentation/filesystems/seq\_file.txt文件

作为linux 自带的关于不同接口的introduce，而且都是作者本人编写，是最正规的说明。

<http://lwn.net/Articles/driver-porting/>

http://lwn.net/Articles/22359/

序言

Kernel 提供了好几种方式，让设备驱动提供信息给用户或者是系统。其中一个比较常用的技术就是创建虚拟文件，比如，debugfs，/proc 或其他地方。

创建虚拟文件仅仅是返回一串字符串，是比较简单的事情。但是，当输出的内容是很复杂的情况时，处理多次读操作时，则需要特别关心 虚拟文件中的读的位置。

然后对虚拟文件的读位置的处理，传统上，内核的许多实现都是错误的。

Seq\_file 机制的接口需要包含<Linux/seq\_file.h>,有三个方面需要注意：

1. 存在一个迭代器，能够让虚拟文件逐步的将内容呈现。
2. 一些用于格式化输出的实用函数不用担心顾虑输出缓存output buffer 的东西。
3. 有个专门操作虚拟文件的file\_operations 的函数集合。

在kernel 3.10 版本之前

entry = create\_proc\_entry("sequence", 0, NULL);

if (entry)

entry->proc\_fops = &ct\_file\_ops;

kernel 3.10 版本之后；一条语句通过。

entry = proc\_create("sequence", 0, NULL, &ct\_file\_ops);

----------------------------------------------------------------------------

entry = proc\_create\_data ("sequence", 0, NULL, &ct\_file\_ops, \*file->private\_data );

多出一个可传递参数；

-----------------------------------------------------------------------------

使用seq\_file 来实现虚拟文件，需要实现一个简单的迭代器对象。用这个迭代器对象来逐步的输出感兴趣的数据。迭代器必须能够移动到特殊的位置，位置的解析也是取决于(is up to) iterator. 无论是何种方式，位置都是对数据的生成有着重要影响的，使用时，应该不需要关心 读指针 翻译成 虚拟文件中的offset。有个例外，就是position 0 应该表明文件的开始。

iterator just uses the count of the next number it will output as its position

迭代器只使用它将输出的下一个数字的计数作为它的位置

有四个函数必须要实现让迭代器工作。

1. Start :拿一个位置当中参数，返回迭代器，从此位置开始读.
2. Next: 移动迭代器到序列中的下一个位置。
3. Stop：迭代器完成的时候，做一个清除的工作clean up.
4. Show: 此函数 格式化被迭代器指向的输出对象。 正常是返回0，

当遇到问题时something went wrong，传递负数错误码给用户空间。Show函数也可以返回SEQ\_SKIP，那么所有的item 输出都将be dropped.

心得：iterator just like pc pointer，in virtual file to read data.

iterator 迭代器的整个数据结构就是一个 loff\_t 类型,存放着当前的位置，并且没有上限，

static const struct seq\_operations ct\_seq\_ops = {

.start = \*\*\_seq\_start,

.next = \*\*\_seq\_next,

.stop = \*\*\_seq\_stop,

.show = \*\*\_seq\_show,

}；

seq\_file 数据结构的私有域将被使用，当更加复杂的应用场景时。

看看另外一个有意思的函数.seq\_printf()，通过start()函数返回迭代器的值是不值得的。

格式化输出：

seq\_file 部分的代码 管理着 迭代器所创建的输出中的定位，并将其放入用户的缓冲区。

目前，我们已经有了一个函数集 使用seq\_file 系统可以产生输出, 但却还没有将file 能让用户看见。

Making it all work

创建虚拟文件时仍需要实现open() 函数—— file\_operations；

Open()函数的主要作用就是hook everything up.

static int gv9531\_proc\_open(struct inode \*inode, struct file \*file)

{

return single\_open(file, gv9531\_seq\_show, PDE\_DATA(inode));

}

static const struct file\_operations ct\_file\_ops = {

.owner = THIS\_MODULE,

.open = ct\_open,

.read = seq\_read,

.llseek = seq\_lseek,

.release = seq\_release

}；

极度简单化版本 /proc 虚拟文件的创建：

对于极度简单的虚拟文件，有个更早的接口，模块仅仅需要定义一个show() 函数，在show()函数中创建所有的输出打印内容。

在file\_operations 中的open() 函数中,调用

int single\_open(struct file \*file, int (\*show)(struct seq\_file \*m, void \*p), void \*data);

-----定义一个 int (\*show)型 的函数指针。

当使用single\_open() 函数时, 编程人员应该使用 single\_release() 函数取代 seq\_release() 在file\_operations 结构体中避免内存泄漏。