<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：

/\*函数seq\_putc用于把一个字符输出到seq\_file文件\*/  
 int seq\_putc(struct seq\_file \*m, char c);  
   
 /\*函数seq\_puts则用于把一个字符串输出到seq\_file文件\*/  
 int seq\_puts(struct seq\_file \*m, const char \*s);  
  
 /\*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\_escape(struct seq\_file \*, const char \*, const char \*);  
   
  
 /\*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\_printf(struct seq\_file \*, const char \*, ...)\_\_attribute\_\_ ((format(printf,2,3)));  
  
 /\*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\*/  
 int seq\_path(struct seq\_file \*, struct vfsmount \*, struct dentry \*, char \*);

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：

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：

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文件并把它的文件操作设置为exam\_seq\_file\_ops即可：

struct proc\_dir\_entry \*entry;

entry = create\_proc\_entry("exam\_seq\_file", 0, NULL);

if (entry)

entry->proc\_fops = &exam\_seq\_file\_ops;

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
2. int exam\_show(struct seq\_file \*p, void \*v)  
    {  
    …  
    }

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。 下面给出了一个使用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。

//kernel module: seqfile\_exam.c

#include <linux/config.h>

#include <linux/module.h>

#include <linux/proc\_fs.h>

#include <linux/seq\_file.h>

#include <linux/percpu.h>

#include <linux/sched.h>

static struct proc\_dir\_entry \*entry;

static void \*l\_start(struct seq\_file \*m, loff\_t \* pos)

{

loff\_t index = \*pos;

if (index == 0) {

seq\_printf(m, "Current all the processes in system:\n"

"%-24s%-5s\n", "name", "pid");

return &init\_task;

}

else {

return NULL;

}

}

static void \*l\_next(struct seq\_file \*m, void \*p, loff\_t \* pos)

{

task\_t \* task = (task\_t \*)p;

task = next\_task(task);

if ((\*pos != 0) && (task == &init\_task)) {

return NULL;

}

++\*pos;

return task;

}

static void l\_stop(struct seq\_file \*m, void \*p)

{

}

static int l\_show(struct seq\_file \*m, void \*p)

{

task\_t \* task = (task\_t \*)p;

seq\_printf(m, "%-24s%-5d\n", task->comm, task->pid);

return 0;

}

static struct seq\_operations exam\_seq\_op = {

.start = l\_start,

.next = l\_next,

.stop = l\_stop,

.show = l\_show

};

static int exam\_seq\_open(struct inode \*inode, struct file \*file)

{

return seq\_open(file, &exam\_seq\_op);

}

static struct file\_operations exam\_seq\_fops = {

.open = exam\_seq\_open,

.read = seq\_read,

.llseek = seq\_lseek,

.release = seq\_release,

};

static int \_\_init exam\_seq\_init(void)

{

entry = create\_proc\_entry("exam\_esq\_file", 0, NULL);

if (entry)

entry->proc\_fops = &exam\_seq\_fops;

return 0;

}

static void \_\_exit exam\_seq\_exit(void)

{

remove\_proc\_entry("exam\_esq\_file", NULL);

}

module\_init(exam\_seq\_init);

module\_exit(exam\_seq\_exit);

MODULE\_LICENSE("GPL");