


```
1. static int traverse(struct seq_file *m, loff_t offset)
2. {
3.     loff_t pos = 0, index;
4.     int error = 0;
5.     void *p;
6.
7.     m->version = 0;
8.     index = 0;
9.     m->count = m->from = 0;
10.    if (!offset) {                                ①
11.        m->index = index;
12.        return 0;
13.    }
14.    if (!m->buf) {                                ②
15.        m->buf = seq_buf_alloc(m->size = PAGE_SIZE);
16.        if (!m->buf)
17.            return -ENOMEM;
18.    }
19.    p = m->op->start(m, &index);                    ③
20.    while (p) {                                    ④
21.        error = PTR_ERR(p);                        ⑤
22.        if (IS_ERR(p))
23.            break;
24.        error = m->op->show(m, p);                    ⑥
25.        if (error < 0)
26.            break;
27.        if (unlikely(error)) {                      ⑦
28.            error = 0;
29.            m->count = 0;
30.        }
```

```

31.         if (seq_overflow(m))                                ⑧
32.             goto Eoverflow;
33.         if (pos + m->count > offset) {                        ⑨
34.             m->from = offset - pos;
35.             m->count -= m->from;
36.             m->index = index;
37.             break;
38.         }
39.         pos += m->count;                                       ⑩
40.         m->count = 0;
41.         if (pos == offset) {                                  (A)
42.             index++;
43.             m->index = index;
44.             break;
45.         }
46.         p = m->op->next(m, p, &index);                        (B)
47.     }
48.     m->op->stop(m, p);                                         (C)
49.     m->index = index;
50.     return error;
51.
52. Eoverflow:                                                  (D)
53.     m->op->stop(m, p);                                         (E)
54.     kvfree(m->buf);
55.     m->count = 0;
56.     m->buf = seq_buf_alloc(m->size <= 1);                    (F)
57.     return !m->buf ? -ENOMEM : -EAGAIN;                       (G)
58. }

```

①

如果traverse到virtual file的首部，实际上没什么好做的。唯一要做的就是设置seq_file->index为0。

这个index也是.start() callback的接受的输入参数

in seq_read()

```
1.      /* we need at least one record in buffer */
2.      pos = m->index;
3.      p = m->op->start(m, &pos);
```

②

如果是真正第一次，则要分配one page buf

③

.start() callback可以做一些枚举item之前的初始化工作，而.stop() callback则可以作与.start()相反的工作。

这里.start()可以修改index的值（所以传入的是pointer）

在traverse()中调用.start()时，传入的index总是0(在seq_read function则未必);在.start()中可以修改index的值，在⑤中seq_file也

会修改index的值！

④

.start()或.next（）callback的返回值应该是某个item的地址，如果为NULL，则结束。

⑤

如果.start()或.next()callback的返回值是负的错误值,则也跳出循环,但.stop() callback还是要运行的。以便作.start()的反工作。

⑥

.show() callback接受的输入参数应该就是.start()或.next()返回的item的指针, seq_file的client可以通过seq_printf()函数往buf中写入data

.show()返回0,自然是出错了,所以要跳出循环

⑦

正常情况下.show()应该返回0,返回正数,???

这里只是把count置零,表示buf中没有内容!

⑧

如果在.show()以后, buf内被填满,自然要去扩大buf

⑨

local variable pos用于跟踪读取到的内容总数,也就是当前virtual file的file pointer(随着不断.show, file pointer在不断往后移动)

if (pos + m->count > offset)

pos + m->count 就是已经读取到的file pointer, 如果大于所要定位到的offset

```
m->from = offset - pos;
```

```
m->count -= m->from;
```

```
m->index = index;
```

???

⑩

还没travser到offset

pos跟踪当前virtual file pointer的移动

```
pos += m->count;
```

```
m->count = 0;
```

这里local variable pos用于追踪m->buf[m->size]中已经fill的内容，即

[m->buf, m->buf + pos]已经被占用。

这里m->count = 0;完全没看懂？而且不是会造成在下一次.show()时，seq_printf()会把内容写到m->buf + m->count = m->buf + 0 吗？这不是把原来[m->buf, m->buf + pos]中的内容给破坏了吗？

(A)

最近一次的.show()后，正好移动到offset

见⑩中，pos中表示m->buf[m->size]中下一个可写入内容的position，如果与offset相等，表示traverse()移动到位了。

```
index++;  
  
m->index = index;  
  
break;
```

从这里看出index作为一个索引一样的东西。假设seq_file要输出的内容都是iterable的，而非stream(流式)的。即item_0, item_1, item_2, ..., item_n

那么这里index作为这些item的索引。

在.start()时，seq_file传入的index为0,这可以理解，告诉seq_file的client这时该输出的是第0份item。.start()可以修改该index值，以便把信息带给.next()。

在.next()时，会把该index也传入，而.next()也可以修改index参数，以便给带给下一次.next()必要的信息。

当然seq_file的client可以不使用该信息。

如果traverse()移动到位了，这里的index也要记录下来，记录在m->index。表示当移动到目前位置时，index的值是多少。

这里之所以要记录该index是为了seq_read()中再次启动

```
item = m->op->start(m->index)
```

```
while(item)
```

```
{
```

```
    m->op->show(item)
```

```
    item = m->op->next(item, m->index)
```

```
}
```

```
m->op->stop(item)
```

如上当m->op->start(m->index)时，可以接着traverse()继续enumerate item下去。

即seq_file.start(m, &index),这里传入的index不一定是0的，反正是告诉seq_file的client应当从哪儿开始。

比如，前面traverse()已经移动到了item_8对应的position，那么seq_read()中的m->op->start(m->index)中的m->index就不是0了，而是9了。

(B)

还未移动到offset，所以还要往后读取

(C)

无论那种情况退出loop，都要调用.stop()。

.start()与.stop()是对称调用的，就象c++中对象的ctor和dtor

m->index = index;

对⑨和(A)而言的退出loop，本行是多余的。

(D)

buf不够了

(E)

要与.start()匹配，必须调用.stop()

(F)

buf扩大为原来一倍

(G)

如果扩大buf失败，返回-ENOMEM

返回-EAGAIN，表示client应该再次调用travser()

```

1.  /**
2.   *   seq_lseek -    ->llseek() method for sequential files.
3.   *   @file: the file in question
4.   *   @offset: new position
5.   *   @whence: 0 for absolute, 1 for relative position
6.   *
7.   *   Ready-made ->f_op->llseek()
8.   */
9.  loff_t seq_lseek(struct file *file, loff_t offset, int whence)
10. {
11.     struct seq_file *m = file->private_data;
12.     loff_t retval = -EINVAL;
13.
14.     mutex_lock(&m->lock);
15.     m->version = file->f_version;
16.     switch (whence) {
17.     case SEEK_CUR:
18.         offset += file->f_pos;
19.     case SEEK_SET: ①
20.         if (offset < 0) ②
21.             break;
22.         retval = offset; ③
23.         if (offset != m->read_pos) { ④
24.             while ((retval = traverse(m, offset)) == -EAGAIN)
25.                 ;
26.             if (retval) { ⑤
27.                 /* with extreme prejudice... */
28.                 file->f_pos = 0;
29.                 m->read_pos = 0;
30.                 m->version = 0;
31.                 m->index = 0;

```

```

32.         m->count = 0;
33.     } else { ⑥
34.         m->read_pos = offset;
35.         retval = file->f_pos = offset;
36.     }
37. } else {
38.     file->f_pos = offset; ⑦
39. }
40. }
41. file->f_version = m->version;
42. mutex_unlock(&m->lock);
43. return retval;
44. }

```

①

从virtual file文件头开始的offset

②

offset < 0自然是invalid

③

正常情况下返回值就是设置的file pointer

④

offset != m->read_pos

表示lseek要移动到的offset与当前seq_file读取后记录的当前file pointer不一致

由于seq_file只能sequence access，所以麻烦大了，通过traverse()把file pointer移动到
到这里指定的offset。

⑤

traverse()正常退出返回0，即seq_file的position已经被移动到要求的position,返回非零，
出错了，所以这里初始化fields

⑥

traverse()正常退出

```
m->read_pos = offset;  
retval = file->f_pos = offset;
```

m->read_pos记录当前seq_file读取到的position

lseek返回移动到的position

⑦

如果lseek request的position等于seq_file当前读取的position，则很简单，只要更新以下file中记录
file pointer

的值即可。

```

1.  /**
2.   *   seq_escape -   print string into buffer, escaping some characters
3.   *   @m:   target buffer
4.   *   @s:   string
5.   *   @esc:  set of characters that need escaping
6.   *
7.   *   Puts string into buffer, replacing each occurrence of character from
8.   *   @esc with usual octal escape.  Returns 0 in case of success, -1 - in
9.   *   case of overflow.
10.  */
11. int seq_escape(struct seq_file *m, const char *s, const char *esc)
12. {
13.     char *end = m->buf + m->size;
14.     char *p;
15.     char c;
16.
17.     for (p = m->buf + m->count; (c = *s) != '\0' && p < end; s++) {
18.         if (!strchr(esc, c)) {
19.             *p++ = c;
20.             continue;
21.         }
22.         if (p + 3 < end) {
23.             *p++ = '\\';
24.             *p++ = '0' + ((c & 0300) >> 6);
25.             *p++ = '0' + ((c & 070) >> 3);
26.             *p++ = '0' + (c & 07);
27.             continue;
28.         }
29.         seq_set_overflow(m);
30.         return -1;
31.     }

```

```
32.         m->count = p - m->buf;  
33.  
         return 0;  
34.     }
```

比如

```
1.     seq_escape(m, dp->format, "\\t\\r\\n\\");
```

把dp->format string中的凡是属于"\\t\\r\\n\\"字符替换成对应字符的文字版的八进制表示。

替换后的string存放到m->buf + m->count开始的space中。

\\n, 0xA, 012

\\t, 0x9, 011

\\r, 0xD, 015

\\", 0x22, 042

假如dp->format = "test string\\n 123\\r ABC\\t"

则替换后的string为

m->buf + m->count = "test string\\012 123\\015 ABC\\011"

m->count += strlen("test string\\012 123\\015 ABC\\011")

```

1.  /**
2.   * seq_path - seq_file interface to print a pathname
3.   * @m: the seq_file handle
4.   * @path: the struct path to print
5.   * @esc: set of characters to escape in the output
6.   *
7.   * return the absolute path of 'path', as represented by the
8.   * dentry / mnt pair in the path parameter.
9.   */
10. int seq_path(struct seq_file *m, const struct path *path, const char *esc)
11. {
12.     char *buf;
13.     size_t size = seq_get_buf(m, &buf);
14.     int res = -1;
15.
16.     if (size) {
17.         char *p = d_path(path, buf, size);
18.         if (!IS_ERR(p)) {
19.             char *end = mangle_path(buf, p, esc);
20.             if (end)
21.                 res = end - buf;
22.         }
23.     }
24.     seq_commit(m, res);
25.
26.     return res;
27. }

```

把struct path *path所代表的path转变成string形式的路径（并escape里面的特定字符）

```
struct path {  
  
    struct vfsmount *mnt;  
  
    struct dentry *dentry;  
  
};
```

```
1.    char *p = d_path(path, buf, size);
```

把struct path表示的path转变成absolute path string

mangle_path() escape path中的指定characters。

seq_file的client最常用的生成内容的函数是seq_printf()

```
1.    int seq_printf(struct seq_file *m, const char *f, ...)  
2.    {  
3.        int ret;  
4.        va_list args;  
5.  
6.        va_start(args, f);  
7.        ret =seq_vprintf(m, f, args);  
8.        va_end(args);  
9.  
10.       return ret;  
11.    }
```



```

1.  int seq_vprintf(struct seq_file *m, const char *f, va_list args)
2.  {
3.      int len;
4.
5.      if (m->count < m->size) {
6.          len = vsprintf(m->buf + m->count, m->size - m->count, f, args);
7.          if (m->count + len < m->size) {
8.              m->count += len;
9.              return 0;
10.         }
11.     }
12.     seq_set_overflow(m);
13.     return -1;
14. }

```

`int vsnprintf(char *str, size_t size, const char *format, va_list ap);`

fill 从 `m->buf + m->count` 开始的 buffer, 并且可 fill 最大空间为 `m->size - m->count`

返回值 `len` 是实际 fill 的 bytes。

`m->count += len;`

更新 `m->buf[m->size]` 中已经被 fill 内容的大小。

如果 `seq_file` 的 client 都通过 `seq_printf` / `seq_vprintf` / `seq_path` 这一类 export functions

来操作 `m->buf` 的话, 就不会使得 `m->count > m->size`, 即 `m->count <= m->size`。