```c
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);
};
```

loff_t *pos and void *v的含义?

---

sample 1

walterzh@walterzh-ThinkPad-T440p:~$ cat /proc/fb

0 inteldrmfb

该信息的输出就利用了seq_file api实现的。

in drivers/video/core/fbmem.c

```c
static const struct file_operations fb_proc_fops = {
        .owner          = THIS_MODULE,
        .open           = proc_fb_open,
        .read           = seq_read,
        .llseek         = seq_lseek,
        .release        = seq_release,
};

proc_create("fb", 0, NULL, &fb_proc_fops);

static const struct seq_operations proc_fb_seq_ops = {
        .start  = fb_seq_start,
        .next   = fb_seq_next,
        .stop   = fb_seq_stop,
        .show   = fb_seq_show,
};

static int proc_fb_open(struct inode *inode, struct file *file)
{
        return seq_open(file, &proc_fb_seq_ops);
}

static void *fb_seq_start(struct seq_file *m, loff_t *pos)
{
        mutex_lock(®istration_lock);
        return (*pos < FB_MAX) ? pos : NULL;
}

static void *fb_seq_next(struct seq_file *m, void *v, loff_t *pos)
{
        (*pos)++;
        return (*pos < FB_MAX) ? pos : NULL;
}

static void fb_seq_stop(struct seq_file *m, void *v)
{
        mutex_unlock(®istration_lock);
}

static int fb_seq_show(struct seq_file *m, void *v)
{
        int i = *(loff_t *)v;
        struct fb_info *fi = registered_fb[i];

        if (fi)
                seq_printf(m, "%d %s\n", fi->node, fi->fix.id);
        return 0;
}
```

/proc/fb就是为了输出如下array中的某些信息。seq_file的.fb_seq_next()就用于enumerate

该array的成员。每次一个。当返回NULL，表示枚举完毕。

struct fb_info *registered_fb[FB_MAX] __read_mostly;

每次structure seq_operations的.next callback和.show callback就是输出上面array的

一个entry。具体.next 负责指向array的下一个entry，而.show负责输出该entry中的相关信息。

而.start和.stop负责枚举array中各个entries前的可能的初始化和收尾工作。

即

```
if(seq.start())
{
   while (seq.next())
         seq.show()
   seq.stop()
}
```

.next callback返回NULL，表示没有东西可以枚举了。

这个sample中loff_t *pos表示array的index。

```
static void *fb_seq_start(struct seq_file *m, loff_t *pos)
```

*pos是0,表示是array的index为0

```
static void *fb_seq_next(struct seq_file *m, void *v, loff_t *pos)
```

*pos是fb_seq_start()或fb_seq_next()返回的*pos值，即*pos完全是由这两个callback修改并解释的。

```
static int fb_seq_show(struct seq_file *m, void *v)

      int i = *(loff_t *)v;
```

*v被解释成了*pos一样?为什么可以？这个sample看不太出来。

static void *fb_seq_next(struct seq_file *m, void *v, loff_t *pos)

static int fb_seq_show(struct seq_file *m, void *v)

这里的void *v可能是fb_seq_start()和fb_seq_next()的返回值。

比如这里fb_seq_start()返回*pos,也就是array的index(0),则在接下来调用fb_seq_next()时

传入的void *v即为fb_seq_start()返回的*pos。同样接下来的fb_seq_next()的输入参数

void *v则是上一个fb_seq_next()的返回值。而fb_seq_show()的输入参数void *v则是fb_seq_next()

的返回值。

---

sample 2

in drivers/gpio/gpiolib.c

```c
static void *gpiolib_seq_start(struct seq_file *s, loff_t *pos)
{
        unsigned long flags;
        struct gpio_chip *chip = NULL;
        loff_t index = *pos;

        s->private = "";

        spin_lock_irqsave(&gpio_lock, flags);
        list_for_each_entry(chip, &gpio_chips, list)
                if (index-- == 0) {
                        spin_unlock_irqrestore(&gpio_lock, flags);
                        return chip;
                }
        spin_unlock_irqrestore(&gpio_lock, flags);

        return NULL;
}

static void *gpiolib_seq_next(struct seq_file *s, void *v, loff_t *pos)
{
        unsigned long flags;
        struct gpio_chip *chip = v;
        void *ret = NULL;

        spin_lock_irqsave(&gpio_lock, flags);
        if (list_is_last(&chip->list, &gpio_chips))
                ret = NULL;
        else
                ret = list_entry(chip->list.next, struct gpio_chip, list);
        spin_unlock_irqrestore(&gpio_lock, flags);
```

```

	s->private = "\n";
	++*pos;

	return ret;
}

static void gpiolib_seq_stop(struct seq_file *s, void *v)
{
}

static int gpiolib_seq_show(struct seq_file *s, void *v)
{
	struct gpio_chip *chip = v;
	struct device *dev;

	seq_printf(s, "%sGPIOs %d-%d", (char *)s->private,
			chip->base, chip->base + chip->ngpio - 1);
	dev = chip->dev;
	if (dev)
		seq_printf(s, ", %s/%s", dev->bus ? dev->bus->name : "no-bus",
			dev_name(dev));
	if (chip->label)
		seq_printf(s, ", %s", chip->label);
	if (chip->can_sleep)
		seq_printf(s, ", can sleep");
	seq_printf(s, ":\n");

	if (chip->dbg_show)
		chip->dbg_show(s, chip);
	else
		gpiolib_dbg_show(s, chip);
```

```c

        return 0;
}

static const struct seq_operations gpiolib_seq_ops = {
        .start = gpiolib_seq_start,
        .next = gpiolib_seq_next,
        .stop = gpiolib_seq_stop,
        .show = gpiolib_seq_show,
};

static int gpiolib_open(struct inode *inode, struct file *file)
{
        return seq_open(file, &gpiolib_seq_ops);
}

static const struct file_operations gpiolib_operations = {
        .owner          = THIS_MODULE,
        .open           = gpiolib_open,
        .read           = seq_read,
        .llseek         = seq_lseek,
        .release        = seq_release,
};

static int __init gpiolib_debugfs_init(void)
{
        /* /sys/kernel/debug/gpio */
        (void) debugfs_create_file("gpio", S_IFREG | S_IRUGO,
                                NULL, NULL, &gpiolib_operations);
        return 0;
}
```

上面的code就是为了输出如下linked-list中的struct gpio_chip中的信息。

LIST_HEAD(gpio_chips);

SoC可能由多个gpio controller，每一个都用struct gpio_chip来表示，并是gpio_chips list

上一个node。code就是枚举这些node。

在Gemstone2 ffc board上，输出如下

```
root@granite2:~# cat /sys/kernel/debug/gpio
GPIOs 0-31, gpio-0:

GPIOs 32-63, gpio-1:
 gpio-34  (mdio-reset          ) out hi

GPIOs 64-95, gpio-2:

GPIOs 96-127, gpio-3:

GPIOs 128-159, gpio-4:

GPIOs 160-191, gpio-5:

GPIOs 192-223, gpio-6:

GPIOs 224-255, gpio-7:

GPIOs 511-511, platform/d4220000.usb3_top, d4220000.usb3_top:
 gpio-511 (vbus                ) in  lo
```

void *gpiolib_seq_start(struct seq_file *s, loff_t *pos)

返回gpio_chips list上第一个struct gpio_chip node，输入参数loff_t *pos应该为0吧？

void *gpiolib_seq_next(struct seq_file *s, void *v, loff_t *pos)

输入参数void *v为gpiolib_seq_start()返回的第一个node或者是前次调用gpiolib_seq_next()

返回的某个node。

其实这里由于是遍历linked-list，所以完全用不到loff_t *pos参数

int gpiolib_seq_show(struct seq_file *s, void *v)

这里输入参数void *v含义同gpiolib_seq_next()一样。