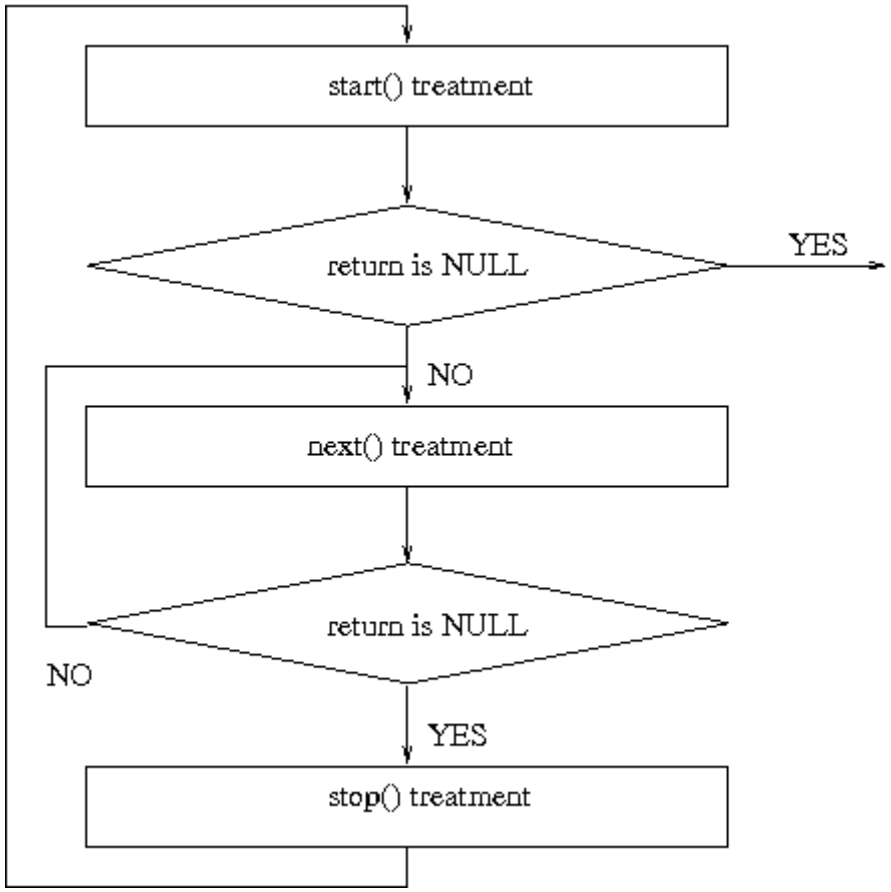
5.4. Manage /proc file with seq_file

As we have seen, writing a /proc file may be quite "complex". So to help people writting /proc file, there is an API named seq_file that helps formating a /proc file for output. It's based on sequence, which is composed of 3 functions: start(), next(), and stop(). The seq_file API starts a sequence when a user read the /proc file.

A sequence begins with the call of the function start(). If the return is a non NULL value, the function next() is called. This function is an iterator, the goal is to go thought all the data. Each time next() is called, the function show() is also called. It writes data values in the buffer read by the user. The function next() is called until it returns NULL. The sequence ends when next() returns NULL, then the function stop() is called.

BE CARREFUL: when a sequence is finished, another one starts. That means that at the end of function stop(), the function start() is called again. This loop finishes when the function start() returns NULL. You can see a scheme of this in the figure "How seq_file works".

Figure 5-1. How seq_file works



Seq_file provides basic functions for file_operations, as seq_read, seq_lseek, and some others. But nothing to write in the /proc file. Of course, you can still use the same way as in the previous example.

Example 5-4. procfs4.c

```
/**
 *  procfs4.c -  create a "file" in /proc
 *          This program uses the seq_file library to manage the /proc file.
 *
 */

#include <linux/kernel.h>  /* We're doing kernel work */
#include <linux/module.h>/* Specifically, a module */
#include <linux/proc_fs.h>/* Necessary because we use proc fs */
#include <linux/seq_file.h>             /* for seq_file */

#define PROC_NAME       "iter"

MODULE_AUTHOR("Philippe Reynes");
MODULE_LICENSE("GPL");

/**
 * This function is called at the beginning of a sequence.
 * ie, when:
 *          - the /proc file is read (first time)
 *          - after the function stop (end of sequence)
 *
 */
static void *my_seq_start(struct seq_file *s, loff_t *pos)
{
            static unsigned long counter = 0;

            /* beginning a new sequence ? */
            if ( *pos == 0 )
            {
                        /* yes => return a non null value to begin the sequence */
                        return &counter;
            }
            else
            {
                        /* no => it's the end of the sequence, return end to stop reading */
                        *pos = 0;
                        return NULL;
            }
}

/**
 * This function is called after the beginning of a sequence.
 * It's called untill the return is NULL (this ends the sequence).
 *
 */
static void *my_seq_next(struct seq_file *s, void *v, loff_t *pos)
{
            unsigned long *tmp_v = (unsigned long *)v;
            (*tmp_v)++;
```

```
        (*pos)++;
        return NULL;
}

/**
 * This function is called at the end of a sequence
 *
 */
static void my_seq_stop(struct seq_file *s, void *v)
{
        /* nothing to do, we use a static value in start() */
}

/**
 * This function is called for each "step" of a sequence
 *
 */
static int my_seq_show(struct seq_file *s, void *v)
{
        loff_t *spos = (loff_t *) v;

        seq_printf(s, "%Ld\n", *spos);
        return 0;
}

/**
 * This structure gather "function" to manage the sequence
 *
 */
static struct seq_operations my_seq_ops = {
        .start = my_seq_start,
        .next  = my_seq_next,
        .stop  = my_seq_stop,
        .show  = my_seq_show
};

/**
 * This function is called when the /proc file is open.
 *
 */
static int my_open(struct inode *inode, struct file *file)
{
        return seq_open(file, &my_seq_ops);
};

/**
 * This structure gather "function" that manage the /proc file
 *
 */
static struct file_operations my_file_ops = {
        .owner   = THIS_MODULE,
        .open    = my_open,
```

```
        .read    = seq_read,
        .llseek  = seq_lseek,
        .release = seq_release
};


/**
 * This function is called when the module is loaded
 *
 */
int init_module(void)
{
        struct proc_dir_entry *entry;

        entry = create_proc_entry(PROC_NAME, 0, NULL);
        if (entry) {
                entry->proc_fops = &my_file_ops;
        }

        return 0;
}

/**
 * This function is called when the module is unloaded.
 *
 */
void cleanup_module(void)
{
        remove_proc_entry(PROC_NAME, NULL);
}
```

If you want more information, you can read this web page:

- http://lwn.net/Articles/22355/
- http://www.kernelnewbies.org/documents/seq_file_howto.txt

You can also read the code of fs/seq_file.c in the linux kernel.