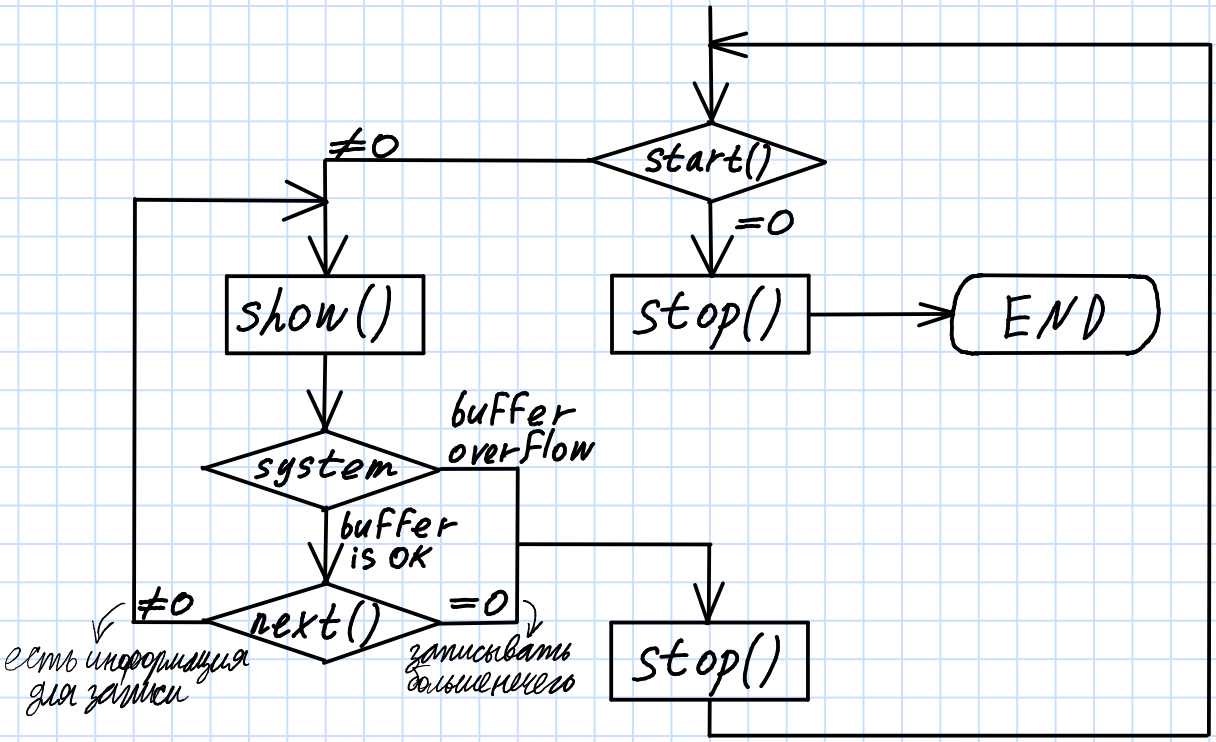
Взаимосвязь ф-ций для работы с sequence файлами



buffer overflow – буфер заполнен и след. запись не м. б. выполнена;
(перепланируй м. б. только попытка");

Варианты решения проблемы:
1) Система может создать доп. буфер (размер равен исходному)
2) Если память выделить невозможно, система может вернуть ошибку;

Ф-ция stop() вызывается в 3 (ситуациях) контекстах:
1) сразу после start() – окончание вывода данных;
2) после show() – буфер заполнен и м. б. создан доп. объём памяти (исх. буфер м. б. расширен на тот же объём, который запрашивался до этого); если памяти нет, будет возвращена ошибка (требуется проверка);

3) после next(), когда закончились данные для вывода (нужно либо завершить работу, либо перейти на вывод другого массива данных);

Интерфейс sequence файлов позволяет передавать данные только из kernel в user;

```
#include <linux/module.h>
#include<linux/proc_fs.h>
#include<linux/seq_file.h>
#define PROC_FILE_NAME "myfile"
static struct proc_dir_entry *proc_file;
static char *str;
static int proc_show(struct seq_file *m, void*v)
{
    int error = 0;
    error = seq_printf(m,"%s\n",str);
    return error;
}
static int proc_my_open(struct inode *inode,
    struct file *file)
{
    return single_open(file,proc_show,NULL);
}
```

ф-ция интерфейса single file

```
static const struct proc_ops proc_fops =
{
    .proc_open = proc_my_open,
    .proc_release = single_release,
    .proc_read = seq_read
};
static int __init proc_init(void)
{
    str = "aaa";
    proc_file = proc_create_data(PROC_FILE_NAME,
        S_IRUGO, NULL, &proc_fops, NULL);
    if (!proc_file)
        return -ENOMEM;
    return 0;
}
static void __exit proc_exit(void)
{
    if (proc_file)
        remove_proc_entry(PROC_FILE_NAME,
            NULL);
}
```

/kernel v.5.16.8+

} предописанные fop или соответств. интерфейса

Single file (subsystem) интерфейс создан для облегчения написания кода, предназначенного для передачи информации из kernel в user;

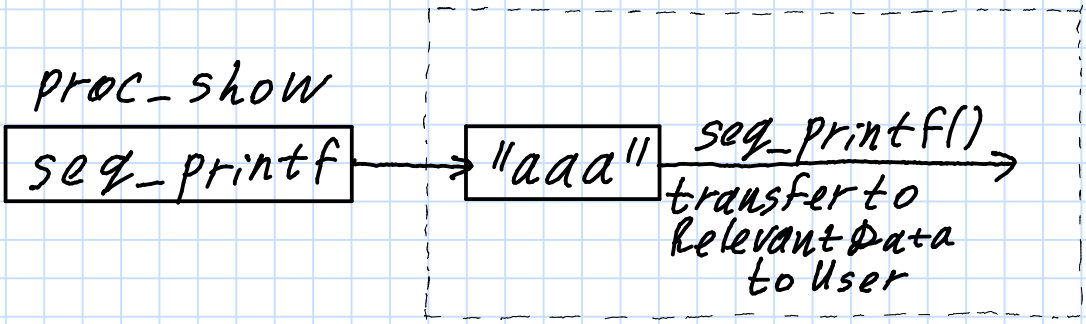
```
int single_open(struct file *file,
    int (*show)(struct seq_file *, void *), void *data)
{
    struct seq_operation *op =
        kmalloc(sizeof(*op), GFP_KERNEL);
    int res = -ENOMEM;
    if (op)
    {
        op->start = single_start; // op - или итератора
        op->stop = single_stop;
        op->next = single_next;
        op->show = show;
        if (!res)((struct seq_file*)file->private_data)
            ->private = data;
        else kfree(op);
    }
    return res;
}
EXPORT_SYMBOL(single_open);
```

open - главная (.) входа;

open → show → seq_printf;

Смысл передачи данных из kernel в user заключается в том, что информация сначала записывается в буфер ядра, и уже из буфера ядра передаётся в user mode или буфер регистра пользователя;

Single file subsystem позволяет передавать до 64 КБ;



Single file subsystem явл. промежуточным слоем отказоустойчивости за счёт снижения опасности обмена данными между пр-вом ядра и пр-вом пользователя „до него - то подобного ф-ции printf()";

Чтобы иметь возможность передавать данные из user в kernel, необходимо написать ф-цию write, в которой вызвать copy_from_user;

При создании файла необх. исп. права доступа S_IRUGO|S_IWUGO;

(начнём с конца)

```
static struct proc_ops proc_fops =
{
    .proc_open = ct_open,
    .proc_read = seq_read,
    .proc_lseek = seq_seek,
    .proc_release = seq_release
};
```

} ф-ции из библиотеки seqfile

```
//Перед структурой должна быть ф-ция
static int ct_open(struct inode *inode,
        struct file *file)
{
        return seq_open(file, &ct_seq_ops);
}
//ф-ции:
static struct seq_operations ct_seq_ops =
{
        .start = ct_seq_start, //Все эти ф-ции
        .next = ct_seq_next, //должны быть описаны
        .stop = ct_seq_stop, //заранее
        .show = ct_seq_show
};
static int ct_seq_show(struct seq_file *s, void *v)
{
        printk(KERN_INFO "In show(), event=%d\n",
                *(int *)v);
        seq_printf(s, "The current value of the event
                number is %d\n", *(int *)v);
        return 0;
}
```

```
static void *ct_seq_start(struct seq_file *s,
    loff_t *pos)
{
    printk(KERN_INFO "Enter start(), pos=%lld,
        seq_file pos=%lu \n", *pos, s->count);
    if (*pos >= limit)
    {
        printk(KERN_INFO "done\n");
        return 0;
    }
    ...
}
```

start() – начало передачи данных;
Указатель pos м.б. инициализирован 0 (начало посл-ти передаваемых данных);
show() – передача данных;
next() – увеличение указателя pos;

```
// Упрощённый вариант:
static void *ct_seq_next(struct seq_file *s,
    void *v, loff_t *pos)
{
    printk(...);
    (*pos)++; // увеличение указателя
    ...
}
```

Основная задача stop() – освобождение памяти;