# Семинар 5 ОС (06.04.2022)

Управление виртуальной памятью поддерживается аппаратно. Но страница загружается в результате страничного прерывания. Нет никакой гарантии что она находится в физической памяти. Поэтому необходимы эти низкоуровневые функции.

Механизм последовательностей (sequence). seq\_file.h

struct seq\_file  
{  
 char \*buf;  
 size\_t size;  
 size\_t from;  
 …  
 loff\_t index;  
 loff\_t read\_pos;  
 …  
 const struct seq\_operations \*op;  
 …  
 const struct file \*file;  
 …  
}

struct seq\_operations  
{  
 void \*(\*start)(strcut 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);  
}

int single\_open(struct file\*, int(\*)(struct seq\_file\*, void\*), void\*);

#include <linux/module.h>  
#include <linux/proc\_fs.h>  
#include <linux/seq\_file.h>  
#define PROC\_FILE\_NAME “Hello”  
static struct proc\_dir\_entry \*proc\_file;  
static char \*out\_str;  
static int proc\_hellow\_show(struct seq\_file \*m, void \*v)  
{  
 int error = 0;  
 error = seq\_printf(m, “%s\n”, out\_str);  
 return error;  
}  
static int proc\_hello\_open(struct inode \*inode, struct file \*file)  
{  
 return single\_open(file, proc\_hellow\_show, NULL); //передаётся указатель на функцию  
}  
static const struct file\_operations proc\_hello\_ops =  
{  
 owner = THIS\_MODULE,  
 open = proc\_hello\_open,  
 release = single\_release;  
 read = seq\_read;  
};

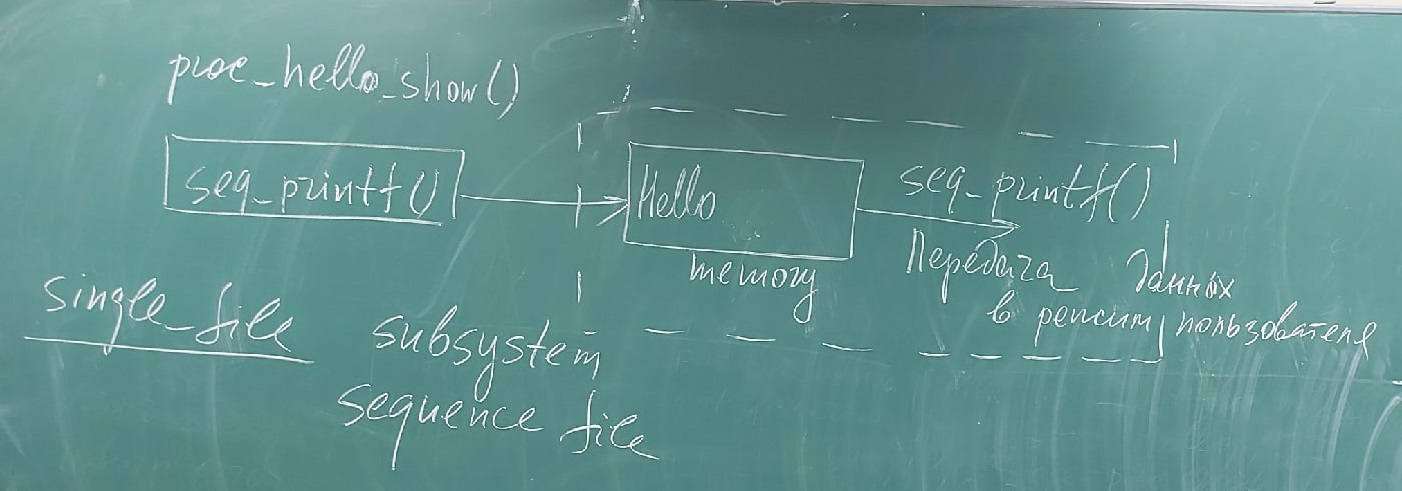
static int \_\_int proc\_hello\_init(void)  
{  
 out\_str = “Hello”;  
 proc\_file = proc\_create\_data(PROC\_FILE\_NAME, S\_IRUGO, NULL, fproc\_hello\_fops, NULL);  
 if (!proc\_file)  
 return -ENOMEM;  
 return 0;  
}

static void \_\_exit proc\_hello\_exit(void)  
{  
 if (proc\_file)  
 remove\_proc\_entry(PROC\_FILE\_NAME, NULL);  
}

!struct proc\_ops!

Разработчики ядра создали промежуточный слой отказоустойчивости для снижения степени сложности обмена данными до привычных функций, например printf.

Базовая идея промежуточного слоя заключается в том что разработчик модуля записывает все данные в достаточно большую область памяти. Рисунок показывает использование функции seq\_print.



proc\_hellow\_show()seq\_printf() **seq\_printf()** --------> **Hello ---------------->** memory Передача данных в режим пользователя

single\_file subsystem представляет собой спец. форму последовательности так называемую sequence\_file. Это подсистема выполняет функции read и release. В результате эти функции могут использоваться без изменений при работе с файлом proc. Для того чтобы создать однофайловый экземпляр модуля, необходимо вызвать функцию single\_open. Эта функция предаст адрес функции которая обычно называется show(). Функция show() передаёт адрес страницы памяти. В приведённом пример данный выводятся при помощи функции seq\_printf(), при этом функция seq\_printf() может быть вызвана произвольное кол-во раз внутри функции show().

У sequence\_file есть ограничения: не предназначены для записи больших объёмов файлов, обычно ограничены 64 Кб.

Функция single\_open получает указатель (то есть адрес) на функцию show.

## Буферизованный и не буферизованный ввод/вывод

Опасность, когда один и тот же файл открывается разными процессами или разными потоками, при этом, необходимо отметить, что то, о чём мы будем говорить является вероятностным событием. Поэтому попробовали открыть один и тот же файл в одной программе. Мы будем делать это в одном процессе и эту же программу, с дополнительным потоком (открываем файл в разных потоках). Кроме этого, надо нарисовать взаимосвязь структур ядра.

struct task\_struct  
{  
 …  
 struct fs\_struct \*fs; //описывает файловую подсистему  
 struct files\_struct \*files; //это таблица открытых файлов процесса  
 …  
}