Задание: Написать функции, посылающие pid пр-сса из user в pid; из kernel в user + 3 поля, посылаемые из task_struct

Семинар 17.04.

Sequence.

· ~~Вирт. файл создаётся в ядре~~. Фактически это буфер. Информацию из буфера ядра мы можем передать циклически с помощью итератора, либо с помощью упрощенного интерфейса. singleshow

· single_show — упрощенный интерфейс.

· Вся информация явл-е последовательность байт — char

linux/seq_file.h

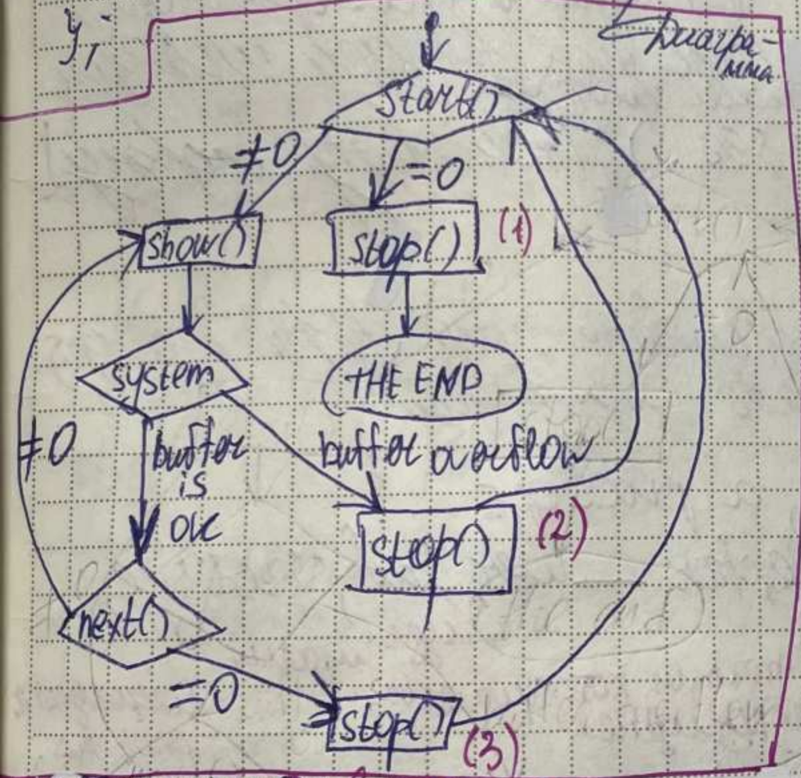
```
struct seq_file
{
    char *buf;  //все инф-ция в ядре явл-е последов-тью байт => char
    size_t size;
    size_t from;
    size_t count;
    size_t pad_until;
    loff_t index;  //очень большой файл
    loff_t read_pos;
    struct mutex lock;
    const struct seq_operations *op;
    int poll_event;
    const struct file *file;  // дескриптор открытых файлов
    void *private;
};
```

loff_t — тип для очень большого объёма информации

у нас вирт. файл, но это открытый файл

В seq_operations определено 4 ф-ции

```
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);
};
```



1) • Если в начале в буфере нет инф-ции, то stop() и конец

2). Если инф-ция есть, то show() (показать)! system - контроль слоттейнов.
2.1) Если возникает ошибка, то stop(). Потом start() и тогда только проверка на NULL.
2.2) Если буфер OK, то next() (итератор). Пока ≠0 show() (показываем). Когда =0 stop() потом не start() и только через stop() выход.

stop() в 3х случаях вызывается:
(1) - окончание работы по распечатке данных
(2) - при выполнении последней операции печати в буфер случилось переполнение страничного буфера и эта операция печати была отменена
(3) - заканчиваеь печать в буфер одних данных и нужно либо завершать работу, либо использ-ть новые данные

. Sequence файлы - файлы последовательностей. Из буфера мы читаем последов-ть байт.

. В ф-ции show() вызывается ф-ция seq_printf код в качестве лабы: текст программы методички по Seq files

указ-ль на parent (тут NULL и файл создается в корневом каталоге /proc)

• В init ф-ции proc_create не возвращает (указатель на proc_dir_entry : proc_create("events", 0, NULL, &ct_file_opens);

имя файла

права доступа (0 - по умолч. установлены)

проинициал. стр-ра file_operations

~~Вм~~ стр-ре file-operations определено поле: owner - всегда пишут THIS_MODULE

!! Здесь нет ф-ции write (она не регистрируется в file-operations) всегда?

• Sequence-files — предназначены для выдачи инф-ции только из ядра в режим пользов. (user.) (т.е. в эту сторону)

Пространство

• Чтобы проинициализировать поле стр-ры proc_ops надо объявить и описать эти ф-ции.

В лабе не file_operations, а proc_ops...?

Внутри seq_read_iter

• ssize_t seq_read(struct file *file, char __user *buf, size_t size, loff_t *ppos)

метод read для seq файла; файл, из кот. читаем; куда читаем; тек позиция

elixir:

• int seq_open(struct file *, const struct seq_operations *);

проинициал. стр-ра

```
Static struct seq-operations ct_seq_ops = {
    .start = ct_seq_start,
    .next = ct_seq_next,
    .stop = ct_seq_stop,
    .show = ct_seq_show,
};
```

even_ptr = (*pos) * 2 (потому что они записывают четное числа)

. Обращаемся к полю count структуры seq_count

. gfp_kernel - выяснить, что это

ct_seq_show():

. ~~[illegible]~~ void*v - это event pointer

m - memory

. struct seq_file - структура, являющаяся дескриптором буфера

ct_seq_next() : - должна изменять указатели

ct_seq_stop():

. Нам важно понять, какие в нашем модуле точки входа, поэтому мы пишем свои функции write, open... ~~В~~ Добавляем printk

В результате мы увидим, когда эти ф-ции вызываются

Аналогичные ф-ции ядра называют Api.

API - это то, что предоставляется юзеру

. Api sequence файла стартует последов-ть, когда польз-ль читает файл /proc

. Последов-ть начинает вызов ф-ции start(). Если она вернула не 0, то вызов next(). next() - итератор, её цель - пройти через все данные. next() приводит к вызову show(). В рез-те данные записываются в буфер польз-ля вызов для чтения.

Будьте внимательны: когда послед-ть заканчивается, другая стартует (в конце stop() или вызыв-ся start())

Семинар, 23.04

Seq файлы

Пишем 3 программы

Singleshow. — упрощенный интерфейс

Упрощенный, т.к. в ~~теле~~ ф-ции single-open инициализ-ие поле стр-ры seq-operations

struct proc_dir_entry — дескриптор файла в вирт. ФС /proc

struct file - дескриптор файла

- proc_create data — возвращает указатель на этот файл

- Если к ф-ции библиотеки не обращается, то этот header не загружается в физ. память.

↑ Это .dll - динамически подгружаемые библиотеки

Пример:

```
#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;
...h_init()
{
    str = "Hello";
    proc_file = proc_create_data(PROC_FILE_NAME
                S_IRUGO, NULL, *proc_fops, NULL);

    if (!proc_file)
        return -ENOMEM;
    return 0;
}
```

необязателен define

этот объявление перемен-ной

указатель на родителя

если файл будет создан в корневом каталоге.

приватные данные (указ-ль void *)

Для этого мы должны определить экземпляр структуры proc-ops.

```
static const struct proc_ops proc_fops = {
        .proc_open = proc_fopen,
        .proc_release = single_release,
        .proc_read = seq_read,
};
```

— это наша ф-ция, она д.б. объявлена и описана выше.

тоже наши ф-ции в них надо вызывать printk чтобы в сис. логе увидеть т. входа

!!! Смысл создания seq файлов: они обеспечивают надежную, аккуратную передачу данных всех записанных в буфер.

```
static int proc_fopen(struct inode *inode,
                      struct file *file)
{
    printk(KERN_INFO "...");
    return single_open(file, proc_fshow, NULL);
}
```

передается ф-ция

указатель на экземпляр seq_operations

```
int single_open(struct file *file, int(*show)(struct
            seq_file*, void *), void *data)
{
    struct seq_operations *op = kmalloc(sizeof(*op),
                          GFP_KERNEL_ACCOUNT);

    int res = -ENOMEN;
    if (op) {
        op->start = single_start;
```

Проход по адресам буфера — переход на след. адрес

```
    op->next = single_next;
    op->stop = single_stop;
    op->show = show;
    res = seq_open(file, op);
    if (!res)
        ((struct seq_file *)(file->private_data)->private =
                                     = data;

    else    kfree(op);
  }
    return res;
}
EXPORT_SYMBOL(single_open);
```

— неразоначивается позже то, в ф-ции single_stop реализовать, ф-ции start, чтобы начать работу со след. последов-тью. Если на выходе start подано null, то вызовется stop

Ф-ция single_start() — определение позиции

- Если мы пишем загр. модуль, то мы можем пользоваться кодом функ generic, кодом функц simple.

- fill_super — ф-ция ядра (нужна для виртуал. ФС)

Основной Закон Программирования:
Ни одна переменная, ф-ция, тип не может использоваться до ее описания, объявления, определения

- single_next() — f+pos (проход по адресам буфера)

```
static char *str;
static int proc_fshow(struct seqfile *m,
                      void *v)
{
    int error = 0;
    error = seq_printf(m, "%s\n", str);
    return error;
}
```

пример на упрощенный интерфейс файлов последовательностей

!!! Ограничение single файлов ~~упрощённого интерфейса~~

- интерфейс single файлов позволяет передавать ограниченный объем данных (≤ 64 Кб) это 16 страниц

- Ф-ция seq_print проверяет заполненность буфера (overflow) (буфер заполнен, если: помимо данных в буфер записать все можно) следующий байт в этот буфер не поместится

При попытке записи данных в заполненный буфер возможно 2 варианта:

1) ошибка (если масштабирование невозможно)

2) система эту ситуацию перехватывает, и будет создан еще один буфер такого же размера

Чтобы в модуль передать идентификатор в ядре — не хватает передачи данных из пр-ва user в пр-во ядра (функции write) proc_write - написать свою ф-цию

Т.е. в struct proc_ops надо включить поле с регистрацией нашей ф-ции write

- 3 варианта - Отличие в размещении интерфейсах
  - Во всех ф-циях пишем printk и выводим только факт обращения: read, write, next, show ..
  - В рез. анализа выводимой в лог информации надо понять, что есть т. входа в модуль

!! • регистрируем все ф-ции.
Мы пишем my_seqread и в нем пишем printk(KERNEL_INFO "read"
"write"
...
...
...

- слов т. входа писать не надо

Лаба по открытым файлам - отчёт бумажный

Лог ошибок: 2 ситуации

- из md2 вызываем local - получаем error
- обоснование порядка загрузки | выгрузки модулей (У Цирюлика написано)

lsmod - вывод к-ва ссылок на модуль

фортунки ✓

seq файлы ✓

- откр. файлы

- VFS

- прерывание со встр. дей-ем (tasklet..) + очереди работ

Проект:
return -1 заменить на return 0

Семинар 30.04.

Буфериз. и небуфериз. вв/вывод

(мало, много открытые файлов)

Сис. вызов open() имеет 2 варианта, т.к. один и т.же сис. вызов может открыть ∃ файл или создать новый файл

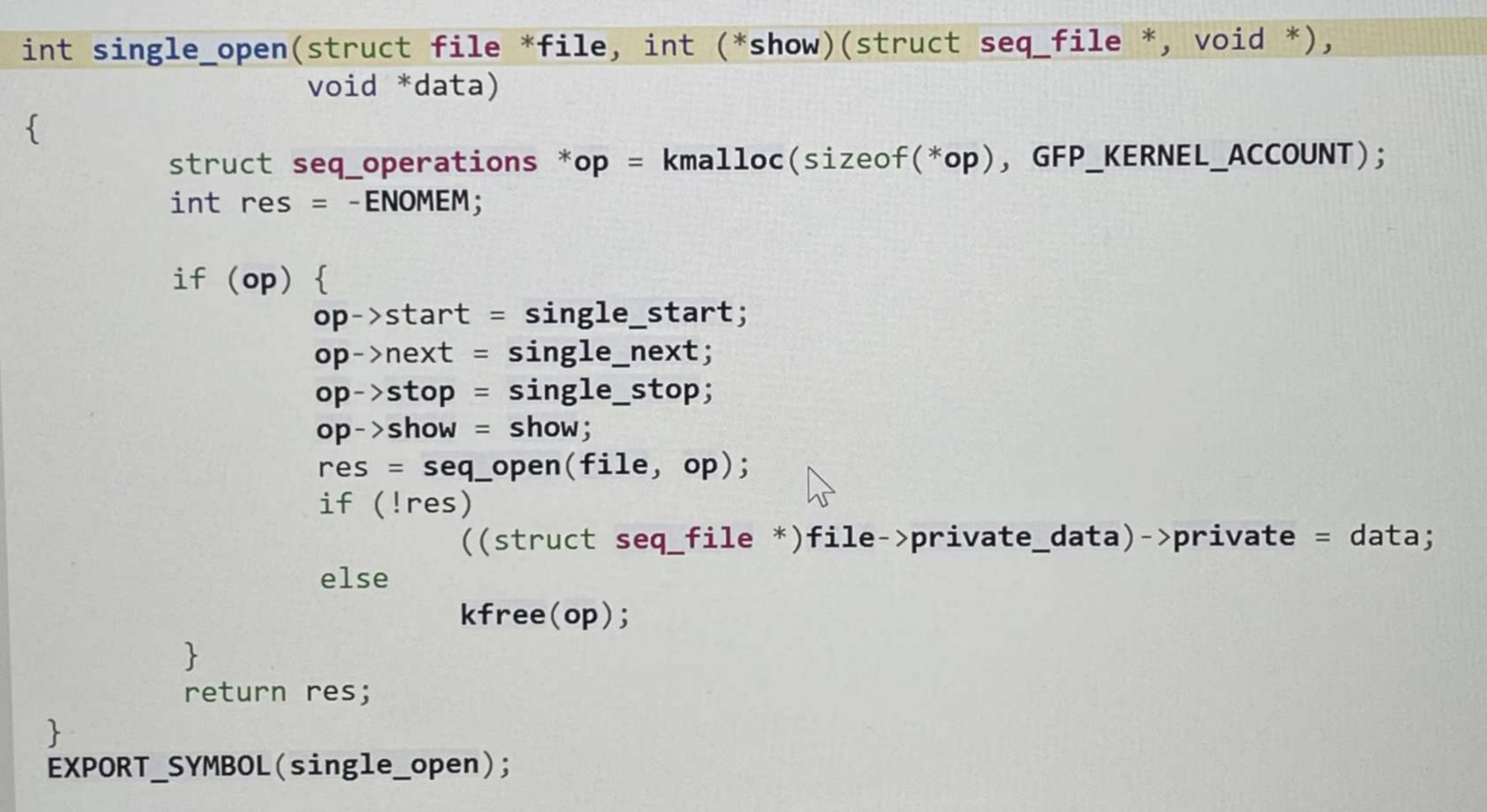
```
int open (const char *pathname, int flags);
int open (const char *pathname, int flags, mode_t mode);
```

Если есть флаг O_CREATE, то д.б. флаг с правами доступа

Флаги: ✓ обязательно используем

O_CREATE
O_EXCL - контроль существование } Эти флаги устанавл. при создании файла

O_EXEC
O_RDONLY
O_RDWR } права доступа
O_APPEND



```
int single_open(struct file *file, int (*show)(struct seq_file *, void *),
                void *data)
{
        struct seq_operations *op = kmalloc(sizeof(*op), GFP_KERNEL_ACCOUNT);
        int res = -ENOMEM;

        if (op) {
                op->start = single_start;
                op->next = single_next;
                op->stop = single_stop;
                op->show = show;
                res = seq_open(file, op);
                if (!res)
                        ((struct seq_file *)file->private_data)->private = data;
                else
                        kfree(op);
        }
        return res;
}
EXPORT_SYMBOL(single_open);
```

Лаб Single:

read before show

read after show

Если потоки detach, то они могут не успеть завершится до конца. Необход-но join, но ∃ несколько вариантов → все содержимое буфера выводится

не верный порядок?

Лаб Single:

Почему интерфейс Single наз. упрощенным?

Мы вырезаем ф-цию

В теле ф-ии single open происходит регистрация ф-ий stop start next (инициализируются после стр-ры seq-operations)

ЗА нас это делается