

Абсолютный адрес - адрес в физической памяти; (лин. адрес - тот, который увеличивается)

Взаимодействие загрузаемых модулей ядра
(по Цирюлину)

Взаимодействие с помощью данных;

md.h:

```
extern char *md1_data;  
extern char *md1_proc(void);
```

md1.c: (модуль, в котором определены данные)

```
#include <linux/init.h>  
#include <linux/module.h>  
#include "md.h"  
MODULE_LICENSE("GPL");
```

```
char *md1_data = "aaa";  
extern char *md1_proc(void)
```

```
{  
    return md1_data;  
}
```

```
static char *md1_local(void)  
{  
    return md1_data;  
}
```

```
static char *md1_noexport(void)  
{  
    return md1_data;  
}
```

→ один макрос и для со-изий, и для данных
EXPORT_SYMBOL(md1_data); // в макросе передаём
EXPORT_SYMBOL(md1_proc); // только имя
// 2 точки в рога в модуль: __init() и __exit():

```
static int __init md_init(void)
{
    printk("+ module md1 start");
    return 0;
}
```

```
static void __exit md_exit(void)
{
    printk("+ stop md1 \n");
}
```

нет уровня про-
токолирования,
чтобы выводить
строки (чтобы
были видны)

есть уровень
по умолчанию?
какой?

```
module_init(md_init);
module_exit(md_exit);
```

md2.c:

```
#include <linux/init.h>
#include <linux/module.h>
#include "md.h"
```

```
MODULE_LICENSE("GPL");
static int __init md_init(void)
```

```
{
    printk("+ md2 start \n");
    printk("+ export md1 data %s \n", md1_data);
    printk("+ export по сути это md1_data proc %s \n", md1_proc());
    return 0;
}
```

```
static void __exit md_exit(void)
{
    printk("+ stop md2 \n");
}
```

```
module_init(md_init);
module_exit(md_exit);
```

m ѓ 3. c :

по все самое, что и в m ѓ 2, только
в init возвр. не 0, а -1;

Этот модуль скормится, но не
загрузится;

При вызове local и поэкспорт в m ѓ 2
возникнут ошибки. Какие?

А если объявить ... local с extern?

Посмотреть все ошибки в syslog (команда
dmesg) и сравнить;

А если загрузить m ѓ 2 до m ѓ 1?

Взаимодействие загр. модулей ядра
можно использовать в курсовой работе.

Следующая лаба: загр. модуль ядра
+
proc

К
У
Р
С
О
В
А
Я

Работа с proc в ядре:
передача из ядра в пользователь
и обратно.

Рассмотрим то, что у user есть proc,
в ядре инициализи все равно больше

copy_to_user, copy_to_kernel, fortunes,
sequence-файлы
↳ передача только kernel → user

Чтобы работать с `proc` в ядре, надо создать в ней файл;

В ядре определена структура

```
<linux/proc_fs.h>
struct proc_dir_entry
{
```

Version 5.16.8
(важно, т.к. эта структура часто переписывается
→ т.к. ядро не стандартизи-
ровано (например, `POST X`)

```
    atomic_t in_use;
    refcount_t refcnt;
    struct list_head pde_ops;
    spin_lock_t pde_unload_lock;
```

любая струк-
тура ядра
содержит
своё св. средство
взаимовоссто-
яния

```
    ...
    const struct inode_operations *proc_iops;
```

```
union
{
```

заменяла `file_ops`,
но не в `union` если в `union`

операции,
определённые
на `inode` ф. с.
`proc`

```
        const struct proc_ops *proc_ops;
        const struct file_operations *proc_dir_ops;
```

```
};
```

```
const struct dentry_operations *proc_dops;
```

```
union
```

```
{
```

исп. для регистрации своих операций
на файле `proc`

```
    const struct seq_operations *seq_ops;
    int (*single_show)(struct seq_file *, void *);
```

```
};
```

```
proc_write_t write;
```

```
void *data;
```

```
unsigned int state_size;
```

```
unsigned int low_ino;
```

```
nlink_t nlink;
```

```
...
```



```

loff_t size;
struct proc_dir_entry *parent;
...
char *name;
u8 flags;
...
}
struct proc_ops {
    unsigned int proc_flags;
    int (*proc_open)(struct inode *, struct file *);
    // в маж. откр. файлов об одном файле порождается
    // только запись, сколько раз он был открыт
    ssize_t (*proc_read)(struct file *, char_user *,
        size_t, loff_t *);
    ...
    ssize_t (*proc_write)(struct file *, const char_user *,
        size_t, loff_t *);
    loff_t (*proc_seek)(struct file *, loff_t, int);
    int (*proc_release)(struct inode *, struct file *);
    // ~ close
    long (*proc_ioctl)(struct file *, unsigned
        int, unsigned long);
}

```

*возможность
определить
действия для работы
с файлами в драйверах*

*функции опреде-
ляют особенности
работы с струк-
турой*

На proc определена функция proc_create_data:
 extern struct proc_dir_entry *
 proc_create_data(const char *, mode_t,
 struct proc_dir_entry *, const struct
 file_operations *, void *);

проверить, можно ли создать proc_ops (в соотв. версиях ядра)

Если же не выполнена проверка - proc_create:
static inline struct proc_dir_entry *
proc_create(const char *name, umode_t mode,
struct proc_dir_entry *parent, const struct
file_operations *proc_fops)
{
return proc_create_data(name, mode, parent,
proc_fops, NULL);
}