Абсолютный адрес – адрес в физической памяти; (лин. адрес – посл. – только увеличивающийся)

Взаимодействие загружаемых модулей ядра (по Цирюлику)

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";
```

→ подтверждение идентификатора

```
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 функций, и для данных

```
EXPORT_SYMBOL(md1_data); //в макросе передаём
EXPORT_SYMBOL(md1_proc); // только имя
//2 точки входа 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");
}
module_init(md_init);
module_exit(md_exit);
```

нет уровня протоколирования, чтобы выделить строки (чтобы были видны)

есть уровень по умолчанию? какой?

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 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);
```

по сути это импорт

md3.c:
то же самое, что и в md2, только в init возвр. не 0, а -1;
Этот модуль скомпилируется, но не загрузится;

При вызове local и noexport в md2 возникнут ошибки. Какие?

А если объявить ___local с extern?

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

А если загрузить md2 до md1?

Взаимодействие загр. модулей ядра можно использовать в курсовой работе.

Следующая лаба: загр. модули ядра + proc

Курсовая

Работа с proc в ядре:
передача из ресурса ядра в ресурс пользователя и обратно.
Несмотря на то, что у user есть proc, в ядре информации всё равно больше

copy_to_user, copy_to_kernel, fortunes ("фортунки"), sequence-файлы
↳передача только kernel→user

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

В ядре определена структура

Version 5.16.8 (важно, т.к. эта структура часто переписывается ⇒ т.е. ядро не стандартизировано (например, POSIX))

```
<linux/proc_fs.h>
struct proc_dir_entry
{
    atomic_t  in_use;
    refcount_t  refcnt;
    struct list_head pde_openers;
    spin_lock_t  pde_unload_lock;
    ...
    const struct inode_operations *proc_iops;
    union
    {
        const struct proc_ops *proc_ops;
        const struct file_operations *proc_dir_ops;
    };
    const struct dentry_operations *proc_dops;
    union
    {
        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;
    ...
```

любая структура ядра содержит собств. средство взаимоисключения

операции, определённые на inode ф. с. proc

заменила file_ops, но тоже есть в union

исп. для регистрации своих операций над файлом proc

```
    loff_t  size;
    struct proc_dir_entry *parent;
     ...
    char *name;
    u8  flags;
     ...
};
```

позволяет определять операции для работы с файлами в драйверах

файлы определяют особенности работы со структурами

```
struct proc_ops
{
    unsigned int proc_flags;
    int (*proc_open)(struct inode *, struct file *);
                     дескриптор файла   дескриптор открытого файла
    //в табл. откр. файлов об одном файле находится
    //столько записей, 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_lseek)(struct file *, loff_t, int);
    int    (*proc_release)(struct inode *, struct file *);
                ~close
     ...
    long (*proc_ioctl)(struct file *, unsigned
            int, unsigned long);
};
```

Для proc определена ф-ция 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, чтобы должна быть proc_ops (в совр. версиях ядра)

Есть более популярная обёртка - 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);
}
```