

День 1. Лекция 2 .

Программирование на уровне ядра.
Модули. Символьные драйверы

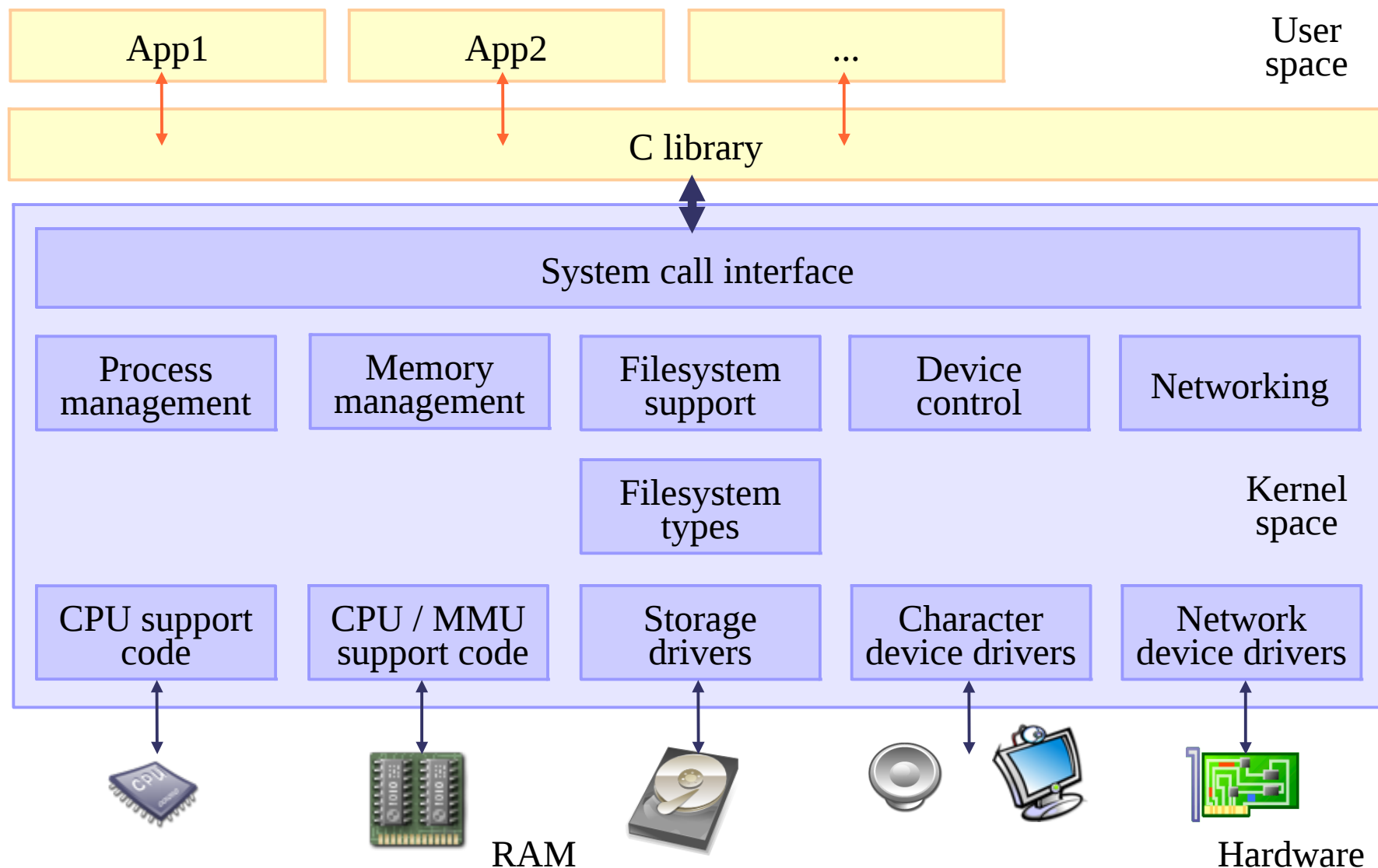
Разработал: Максимов А.Н.

Версия 1.5. 06.2020

Содержание

- Типы драйверов
- Модули ядра
- Передача параметров в модуль
- Внутренняя структура модуля
- Символьный драйвер

Часть 1. Архитектура ядра




Исходные код ядра на www.kernel.org

The Linux Kernel Archive x +


← → ↻ 🏠 <https://www.kernel.org>

The Linux Kernel Archives

About Contact us FAQ Releases Signatures Site news



Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

Latest Release
5.7.7 

mainline:	5.8-rc4	2020-07-05	[tarball]	[patch]	[inc. patch]	[view diff]	[browse]		
stable:	5.7.7	2020-06-30	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
stable:	5.6.19 [EOL]	2020-06-17	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
stable:	3.16.85 [EOL]	2020-06-11	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	5.4.50	2020-06-30	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	4.19.131	2020-07-01	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	4.14.187	2020-06-30	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	4.9.229	2020-06-30	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	4.4.229	2020-06-30	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
linux-next:	next-20200703	2020-07-03						[browse]	

Как скачать исходный код.

➤ Скачать все ядро:

<https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.2.2.tar.gz>

Index of /pub/linux/kernel/ X +

< > ↺ || 🔒 mirrors.edge.kernel.org/pub/linux/kernel/v5.x/

linux-5.2.17.tar.sign	21-Sep-2019 05:28	989
linux-5.2.17.tar.xz	21-Sep-2019 05:28	102M
linux-5.2.18.tar.gz	01-Oct-2019 07:07	157M
linux-5.2.18.tar.sign	01-Oct-2019 07:07	989
linux-5.2.18.tar.xz	01-Oct-2019 07:07	102M
linux-5.2.19.tar.gz	05-Oct-2019 11:19	157M
linux-5.2.19.tar.sign	05-Oct-2019 11:19	989
linux-5.2.19.tar.xz	05-Oct-2019 11:19	102M
linux-5.2.2.tar.gz	21-Jul-2019 07:07	157M
linux-5.2.2.tar.sign	21-Jul-2019 07:07	987
linux-5.2.2.tar.xz	21-Jul-2019 07:07	102M
linux-5.2.20.tar.gz	07-Oct-2019 17:07	157M
linux-5.2.20.tar.sign	07-Oct-2019 17:07	989
linux-5.2.20.tar.xz	07-Oct-2019 17:07	102M
linux-5.2.21.tar.gz	11-Oct-2019 16:31	157M
linux-5.2.21.tar.sign	11-Oct-2019 16:31	989
linux-5.2.21.tar.xz	11-Oct-2019 16:31	102M
linux-5.2.3.tar.gz	26-Jul-2019 07:17	157M
linux-5.2.3.tar.sign	26-Jul-2019 07:17	987
linux-5.2.3.tar.xz	26-Jul-2019 07:17	102M
linux-5.2.4.tar.gz	28-Jul-2019 06:32	157M
linux-5.2.4.tar.sign	28-Jul-2019 06:32	987

Структура исходных кодов ядра (1)

arch/<arch>	Код специфичный для архитектуры
block/	Код подсистемы блочных устройств
COPYING	Условия лицензирования (GNU GPL)
CREDITS	Основные разработчики
crypto/	Криптографические библиотеки
Documentation/	Документация на ядро
drivers/	Драйвера (net, usb, pci...)
firmware/	Различные прошивки
fs/	Файловые системы (fs/ext3/, etc.)
include/	Заголовочные файлы ядра
include/asm<arch>	Архитектурно зависимые заголовочные файлы
include/linux	Заголовочные файлы для Linux kernel
init/	Код инициализации Linux (including main.c)
ipc/	Код IPC

Структура исходных кодов ядра (2)

Kbuild, Kconfig	Часть системы сборки ядра
kernel/	Linux kernel core
lib/	Различные библиотеки (zlib, crc32...)
MAINTAINERS	Список МАНТЕЙНЕРОВ. !!!!
Makefile	Linux makefile (sets arch and version)
mm/	Система управления памятью
net/	Сетевая подсистема
README	Обзор и инструкции по сборке
REPORTINGBUGS	Как отправить Bug report
samples/	Примеры использования
scripts/	Scripts for internal or external use
security/	Реализация модели безопасности (SELinux...)
sound/	Звуковая подсистема и драйвера
Tools/	Дополнительные утилиты (напр. тесты производительности)

Извлечение кода ядра linux

```
$cd /usr/src/kernels
```

```
$mkdir linux-4.1-demo
```

```
$cd linux-4.1-demo
```

Скопировать код ядра linux-4.1.tar.gz

```
tar xzvf linux-4.1.tar.gz
```

 или

```
(tar xfvj linux-4.1.tar.bz2)
```

Скопировать патч patch-4.1.24.gz

```
gunzip patch-4.1.24.gz
```

```
cd linux-4.1
```

```
patch -p1 <../patch-4.1.24
```

Можно отменить патч опция -R

Подготовка к компиляции

Идентифицировать сборку.

Linux-4.1\Makefile

VERSION = 4

PATCHLEVEL = 1

SUBLEVEL = 0

EXTRAVERSION =

NAME = Hurr durr I'ma sheep

Для предотвращения искажения существующих модулей ядра можно указать уникальный идентификатор в EXTRAVERSION. В этом случае модули ядра будут помещаться в /lib/modules/\$VERSION.\$PATCHLEVEL.\$SUBLEVEL-\$EXTRAVERSION.

Сохранить config

cd linux

cp .config config.sav

Настройка ядра

Настройка ядра

```
make mrproper
```

```
make menuconfig
```

Компиляция

```
make
```

Установка

```
sudo make install
```

```
sudo make modules_install
```

make menuconfig

```
.config - Linux/x86 4.16.8-gentoo Kernel Configuration

Linux/x86 4.16.8-gentoo Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module capable

Gentoo Linux --->
[*] 64-bit kernel
General setup --->
[*] Enable loadable module support --->
-* Enable the block layer --->
Processor type and features --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Executable file formats / Emulations --->
[*] Networking support --->
Device Drivers --->
Firmware Drivers --->
File systems --->
Kernel hacking --->
Security options --->
-* Cryptographic API --->
[*] Virtualization --->
Library routines --->

<Select> < Exit > < Help > < Save > < Load >
```

Результаты make

В результате запуска команды `make` создается `vmlinux` - Raw Linux kernel image, не сжатый.

`vmlinux` - zlib compressed kernel image

`arch/<arch>/boot/zImage` (image по умолчанию для arm)

`arch/<arch>/boot/bzImage` (image по умолчанию для i386)

Caution: bz means "big zipped" but not "bzip2 compressed"!

(bzip2 сжатие поддерживается только на i386. Не очень подходит для встраиваемых приложений т.к. Требуется 1 MB RAM для распаковки).

Результаты make-install

`/boot/vmlinuz<version>`

Compressed kernel image. Same as the one in `arch/<arch>/boot`

`/boot/System.map<version>`

Stores kernel symbol addresses

`/boot/initrd<version>.img` (when used by your distribution)

Initial RAM disk, storing the modules you need to mount your root filesystem. `make install` runs `mkinitrd` for you!

`/boot/grub/menu.lst` or `/etc/lilo.conf`

`make install` updates your bootloader configuration files to support your new kernel! It reruns `/sbin/lilo` if LILO is your bootloader

Результаты make modules-install

Файлы модулей ядра имеет расширение .ko (Kernel Object).

/lib/modules/<version>/

-modules.dep – Файл зависимостей модулей

-modules.symbols - Файлы с таблицей символов модуля

Особенности программирования на уровне ядра

Нет функций библиотеки `libc!!!`

Вместо `printf`, надо использовать `printk`

Некоторые важные функции реализованы на уровне ядра

Нет защиты памяти

Небольшой фиксированный размер стека

Возникают вопросы с синхронизацией и конкурентным выполнением задач (к модулю могут обратиться различные процессы, могут возникнуть прерывания, существуют механизмы ядра, которые выполняются асинхронно)

Часть 2. Определение модуля

Модуль — фрагмент кода, который может быть загружен или выгружен из ядра по запросу. Модули расширяют функции системы без необходимости перезагрузки ядра.

Просмотреть список модулей:

lsmod

Установить модуль

insmod module.ko

modprobe module

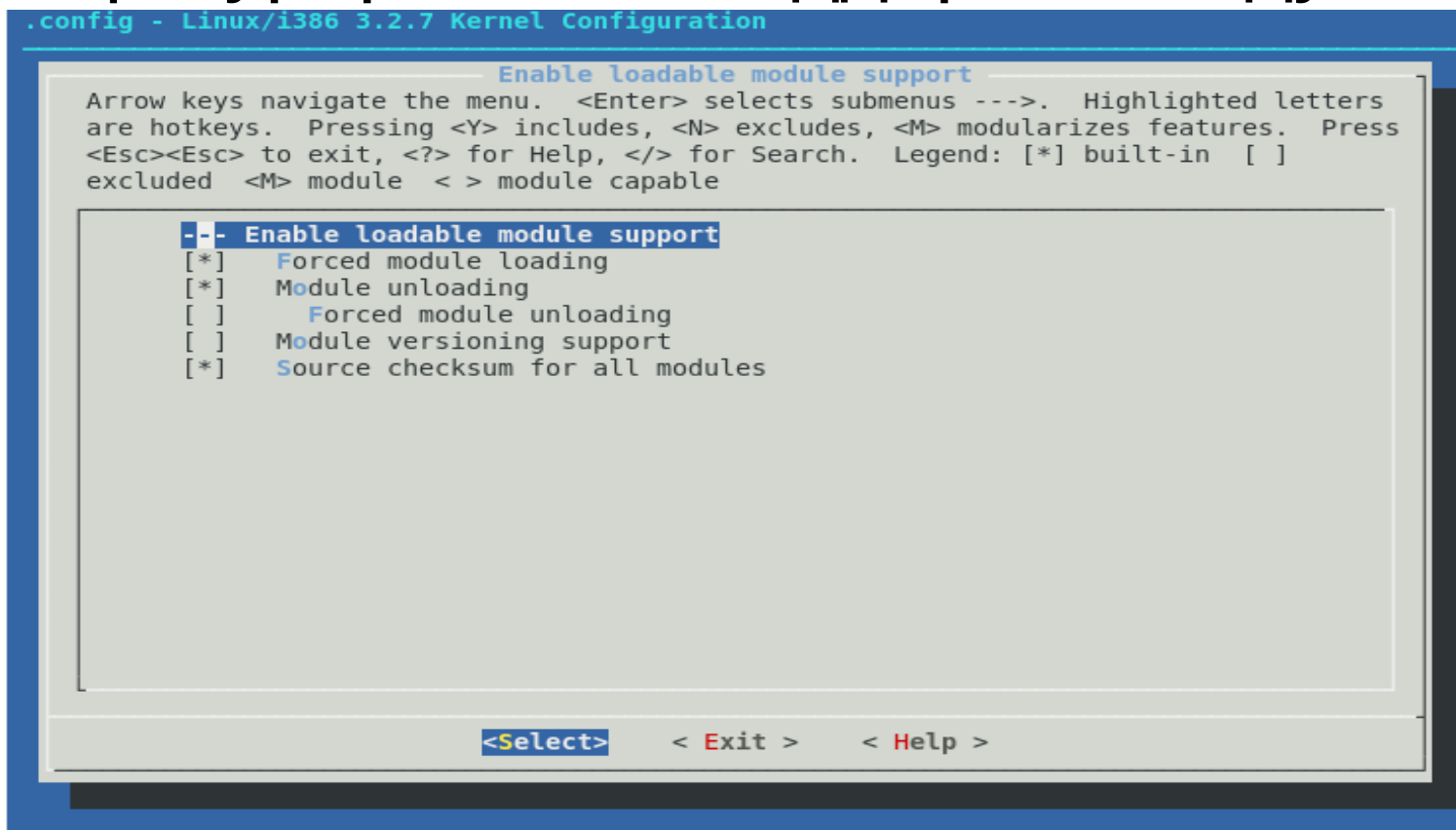
Удалить модуль

rmmod module.ko

Module dependency

depmod module.ko

Конфигурирование поддержки модулей



- Поддержка модулей может быть выключена для повышения безопасности
- Использование модулей теоретически может приводить к фрагментации памяти

Пример модуля

```
#include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_INFO */
MODULE_LICENSE("GPL");

int init_module(void) {
    printk(KERN_INFO "Hello world 1.\n");
    /*
     * Модуль должен возвращать 0 в случае успешной загрузки.
     */
    return 0;
}

void cleanup_module(void) {
    printk(KERN_INFO "Goodbye world 1.\n");
}
```

Компиляция модуля. Makefile

Для компиляции компонентов ядро Makefile должен отличаться от обычного.
Пример:

```
obj-m += hello-1.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Что нужно для сборки на Devnian/Ubuntu

Для компиляции компонентов ядра нужны заголовочные файлы ядра.

```
sudo apt update
```

```
sudo apt install linux-headers-$(uname -r)
```

Метаданные и макросы

Добавление информации о модуле

```
MODULE_AUTHOR(author);
```

```
MODULE_DESCRIPTION(description);
```

```
MODULE_VERSION(version_string);
```

```
MODULE_LICENSE(license);
```

Посмотреть параметры модуля можно так:

```
modinfo helloworld.ko
```

и др.

Определение входной и выходной точек модуля:

```
module_init(init_function);
```

```
module_exit(exit_function);
```

Утилита modinfo

Программа для просмотра информации о модуле Linux

```
[root@lab]# modinfo module5.ko
```

```
filename: module5.ko
```

```
description: This module is a homework
```

```
license: GPL
```

```
author: Ivanov Ivan
```

```
srcversion: 7C328D0F123A8B302DDCD43
```

```
depends:
```

```
vermagic: 2.6.37.emb SMP mod_unload 686
```

Возможные типы лицензий

`include/linux/module.h`

- **GPL**
GNU Public License v2 or later
- **GPL v2**
GNU Public License v2
- **GPL and additional rights**
- **Dual MIT/GPL**
GNU Public License v2 or MIT
- **Dual BSD/GPL**
GNU Public License v2 or BSD
- **Dual MPL/GPL**
GNU Public License v2 or Mozilla
- **Proprietary**
Non free products

printk — печать сообщений

Прототип функции (include/linux/printk.h):

```
int printk(const char * fmt, ...)
```

Поддерживаемые форматы можно посмотреть в Documentation/printk-formats.txt

Сообщения printk имеют приоритет:

```
#define KERN_EMERG    "<0>" /* system is unusable */
#define KERN_ALERT    "<1>" /* action must be taken immediately */
#define KERN_CRIT     "<2>" /* critical conditions */
#define KERN_ERR      "<3>" /* error conditions */
#define KERN_WARNING  "<4>" /* warning conditions */
#define KERN_NOTICE   "<5>" /* normal but significant condition */
#define KERN_INFO     "<6>" /* informational */
#define KERN_DEBUG    "<7>" /* debug-level messages */
```

Пример:

```
printk(KERN_INFO "fb%d: %s frame buffer device\n", info->node,info->fix.id);
```


Просмотр сообщений от printk

- Сообщения помещаются в циркулярный буфер
- Посмотреть можно разными способами:

1) Посмотреть файл

`/var/log/messages`

Файл формируется демоном

`syslogd / klogd`

Посмотреть можно так:

`tail -f`

`/var/log/messages`

Для контроля роста
файла можно использовать
`logrotate`

➤ `cat /proc/kmsg`

➤ **`dmesg`** (“**diagnostic message**”)
Отображает содержимое kernel
log buffer

Просмотр на экране

Сообщения помещаются в циркулярный буфер.

Изменить границу приоритета для отображаемых на экране сообщений можно в файле `/etc/sysctl.conf`

```
kernel.printk = 4 4 1 7
```

Расшифровка значений:

- `console_loglevel`: сообщения с приоритетом выше чем у указанного печатаются на консоле
- `default_message_level`: сообщения без приоритета будут иметь данный приоритет
- `minimum_console_loglevel`: минимальное(highest) значение в которое может быть установлено `console_loglevel`
- `default_console_loglevel`: значение по умолчанию для `console_loglevel`

(дополнительно можно посмотреть <http://linux.die.net/man/2/syslog>)

Для динамического изменения можно:

```
echo "7 1 1 7" > /proc/sys/kernel/printk
```

посмотреть текущее состояние можно так:

```
cat /proc/sys/kernel/printk
```

Часть 3. Передача параметров в модуль

```
#include <linux/moduleparam.h>  
module_param(variable, type, perm);
```

variable — имя переменной

type - тип (можно посмотреть в linux/moduleparam.h)

perm — права доступа (можно посмотреть в linux/stat.h)

Параметр можно изменить из sysfs

Макрос для modinfo:

```
MODULE_PARM_DESC(debug_enable, "Enable deb mode");
```

Пример передачи параметров

```
#include <linux/init.h>

#include <linux/module.h>

#include <linux/moduleparam.h>

MODULE_LICENSE("GPL");

static char *whom = "world";

module_param(whom, charp, 0);

static int howmany = 1;

module_param(howmany, int, 0);          // Передается пара параметров - сколько раз и кому передается привет

static int __init hello_init(void) {
    int i;
    for (i = 0; i < howmany; i++)
        printk(KERN_ALERT "(%d) Hello, %s\n", i, whom);
    return 0;
}

static void __exit hello_exit(void) {
    printk(KERN_ALERT "Goodbye, cruel %s\n", whom);
}

module_init(hello_init);

module_exit(hello_exit);
```

Запуск модуля с параметрами

- Вызов `insmod`:

```
sudo insmod ./hello_param.ko howmany=2 whom=universe
```

- Вызов `modprobe`:

Set parameters in `/etc/modprobe.conf` or in any file in `/etc/modprobe.d/`:

```
options hello_param howmany=2 whom=universe
```

- Случай статической линковки модуля к ядру:

```
options hello_param.howmany=2 hello_param.whom=universe
```

Стиль кодирования

/usr/src/linux/Documentation/CodingStyle

Ключевые особенности:

- Избегать трюков в выражениях
- Размер строки 80 символов
- Разбивать длинные выражения
- Открывающая скобка блока — последняя в строке
- В функциях открывающая скобка с начала строки
- В if скобки не нужны только если и в if и в else по одному выражению
- В указателях * ближе к переменной, чем к типу.
- Отступ в 1 символ вокруг бинарных операций
- Именованное с использованием нескольких слов осуществляется с помощью _
- EXPORT_SYMBOL пишется сразу после функции, которую он экспортирует
- Goto используется для обработок ошибок
- Комментарии C99 не используются

Стиль кодирования

Пример:

```
/*  
 *   This function does ...  
 */  
  
int get_user_id(struct user *u)  
{  
    if (u->condition) {  
        do_smth();  
        do_smth_again();  
    }  
    else {  
        goto error;  
    }  
  
    /* ... */  
}  
  
EXPORT_SYMBOL(get_user_id);
```

Динамическое выделение памяти

malloc - не существует.

kmalloc/kfree выделяет до 128k физической памяти в последовательном блоке

kcalloc(. . .) аналогично kmalloc, но память обнуляется

```
void *kmalloc (size_t size, int flags);
```

flags определены в <linux/mm.h>

GFP_USER associated userspace process sleeps until free memory available

GFP_KERNEL associated kernel function sleeps until free memory available

GFP_ATOMIC doesn't sleep (used in ISRs)

Часть 3. Какой формат имеет модуль ядра?

Для определения формата модуля ядра можно воспользоваться командой файл:

```
[root@trening moduletest1]# file module1.ko
```

```
module1.ko: ELF 32-bit LSB relocatable, Intel 80386, version 1  
(SYSV), not stripped
```

Еще один пример модуля ядра

```
#include <linux/module.h>
```

```
#include <linux/init.h>
```

```
#include <linux/kernel.h>
```

```
int __init init_module(void)
```

```
{
```

```
    printk(KERN_INFO "Hello world 1.\n");
```

```
    /* При ошибке надо вернуть не 0 */
```

```
    return 0;
```

```
}
```

```
void cleanup_module(void)
```

```
{
```

```
    printk(KERN_INFO "Goodbye world 1.\n");
```

```
}
```

Таблица СИМВОЛОВ модуля

Таблицы символом модуля содержатся в следующих сегментах:

__kstrtab – имена символов

__ksymtab – адреса символов, доступных всем типам модулей

__ksymtab_gpl - адреса символов, доступных модулям с лицензией GPL

Для экспорта символом из модуля можно использовать макросы:

EXPORT_SYMBOL

EXPORT_SYMBOL_GPL

Пример (linux-xxx\drivers\net\mii.c):

```
int mii_link_ok (struct mii_if_info *mii) {....}
```

```
EXPORT_SYMBOL(mii_link_ok);
```

Что может показать objdump

objdump –section-headers module1.ko

module1.ko: file format elf32-i386

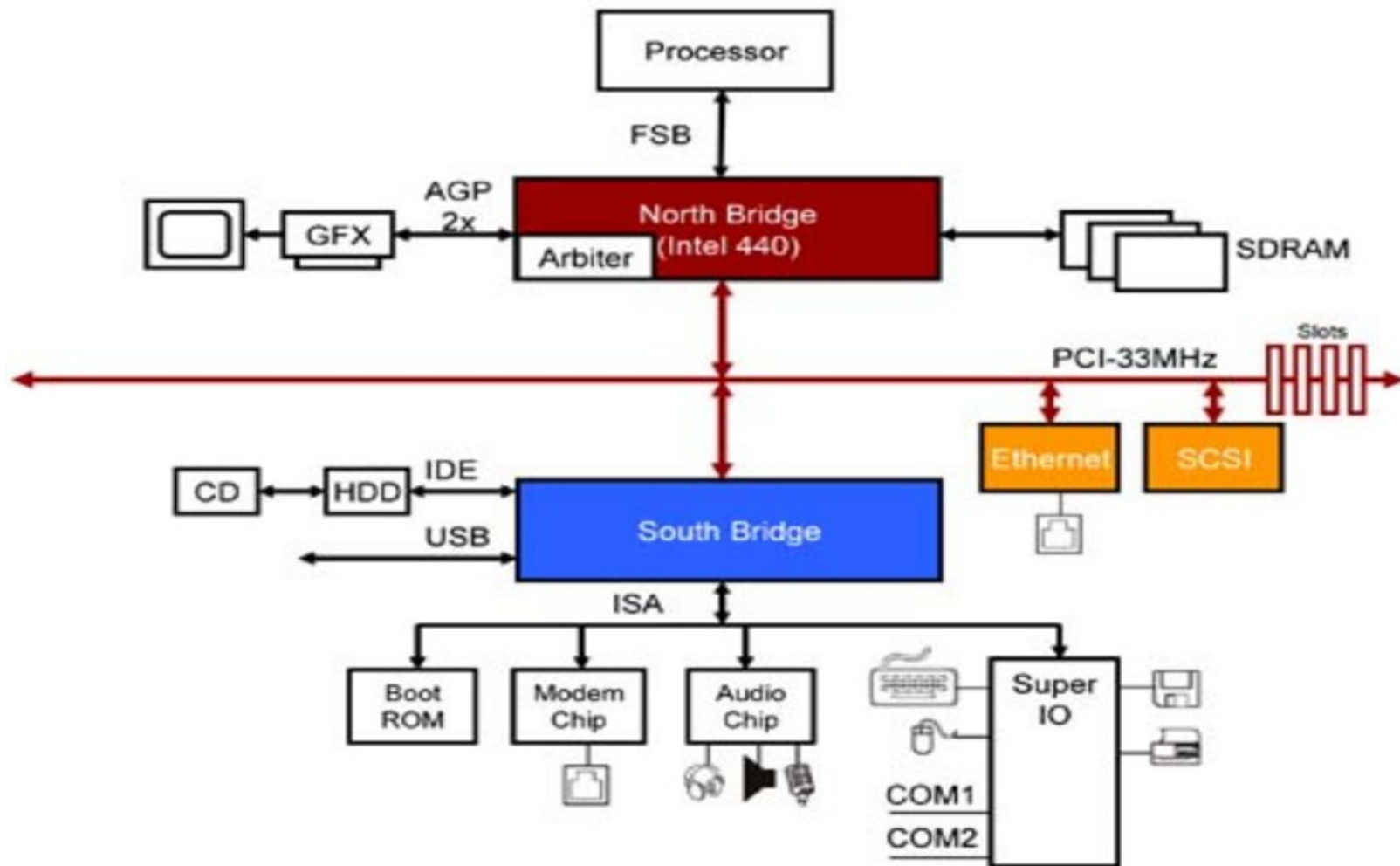
Sections:

Idx	Name	Size	VMA	LMA	File off	Algn
0	.note.gnu.build-id	00000024	00000000	00000000	00000034	2**2
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
1	.text	00000018	00000000	00000000	00000058	2**2
	CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE					
2	.init.text	00000011	00000000	00000000	00000070	2**0
	CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE					
3	.rodata.str1.1	00000028	00000000	00000000	00000081	2**0
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
4	__mcount_loc	00000004	00000000	00000000	000000ac	2**2
	CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA					
5	.modinfo	00000064	00000000	00000000	000000b0	2**2
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
6	.data	00000000	00000000	00000000	00000114	2**2
	CONTENTS, ALLOC, LOAD, DATA					
7	.gnu.linkonce.this_module	00000164	00000000	00000000	00000114	2**2
	CONTENTS, ALLOC, LOAD, RELOC, DATA, LINK_ONCE_DISCARD					
8	.bss	00000000	00000000	00000000	00000278	2**2
	ALLOC					
9	.note.GNU-stack	00000000	00000000	00000000	00000278	2**0
	CONTENTS, READONLY, CODE					
10	.comment	0000005a	00000000	00000000	00000278	2**0
	CONTENTS, READONLY					

Структура module

```
struct module {
    enum module_state state;
    struct list_head list; /* Member of list of modules */
    char name[MODULE_NAME_LEN]; /* Unique handle for this module */
    struct module_kobject mkobj; /* Sysfs stuff. */
    struct module_attribute *modinfo_attrs;
    const char *version;
    const char *srcversion;
    struct kobject *holders_dir;
    const struct kernel_symbol *syms; /* Exported symbols */
    const unsigned long *crcs;
    unsigned int num_syms;
    struct kernel_param *kp; /* Kernel parameters. */
    unsigned int num_kp;
    unsigned int num_gpl_syms; /* GPL-only exported symbols. */
    const struct kernel_symbol *gpl_syms;
    const unsigned long *gpl_crcs;
    unsigned int num_exentries; /* Exception table */
    struct exception_table_entry *extable;
    int (*init)(void); /* Startup function. */
    void *module_init; /* If this is non-NULL, vfree after init() returns */
    void *module_core; /* Here is the actual code + data, vfree'd on unload. */
    unsigned int init_size, core_size; /* Here are the sizes of the init and core sections */
    unsigned int init_text_size, core_text_size; /* The size of the executable code in each section. */
    struct mod_arch_specific arch; /* Arch-specific module values */
    unsigned int taints; /* same bits as kernel:tainted */
    void *percpu; /* Per-cpu data. */
    char *args; /* The command line arguments (may be mangled). People like keeping pointers to this stuff */
#ifdef CONFIG_MODULE_UNLOAD
    struct list_head modules_which_use_me; /* What modules depend on me? */
    struct task_struct *waiter; /* Who is waiting for us to be unloaded */
    void (*exit)(void); /* Destruction function. */
    local_t ref;
#endif
};
```

Аппаратное обеспечение



Типы драйверов

- Символьные драйверы
- Блочные драйверы
- Сетевые драйверы

Символьный драйвер

Символьный драйвер обрабатывает поток байт. Обычно драйвер отвечает за реализацию следующих системных вызовов open, close, read, and write .

Примеры:

console /dev/console

serial ports /dev/ttyS0

IrDA

Bluetooth

...

Доступ к символьным устройствам осуществляется через файловую систему. Они представляются в виде файла.

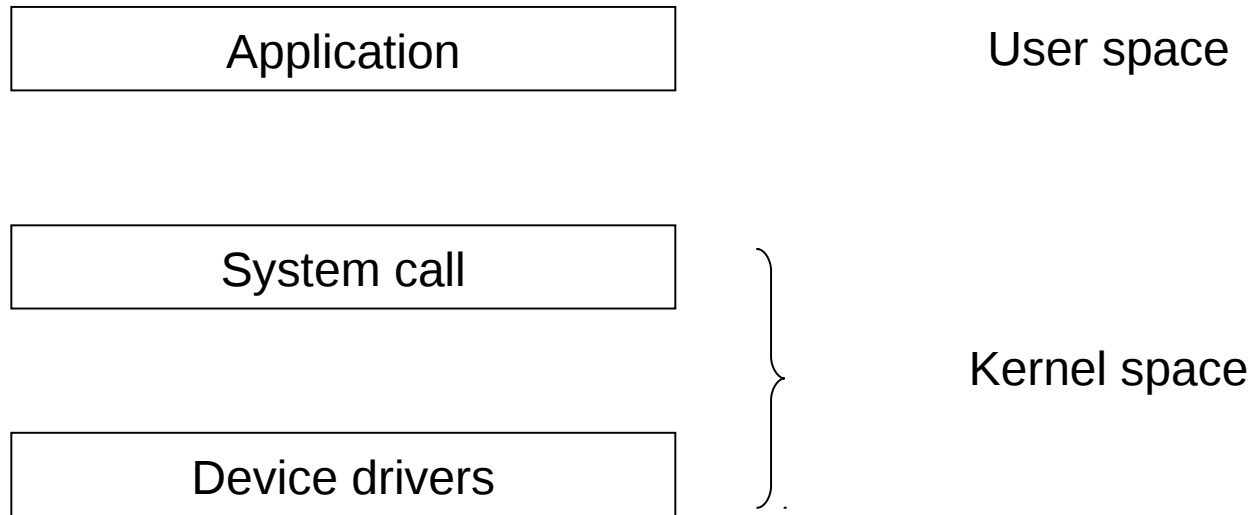
Блочное устройство

- Блочное устройство может иметь файловую систему
- Как и символьное устройство блочное представляется файлом в каталоге /dev
- Блочное устройство оперирует блоками информации (обычно по 512 байт)
- В linux оно может работать с блоками любого размера

Сетевой драйвер

- Служит для обмена информацией с сетевым устройством.
- Обрабатывает пакеты
- Интерфейс сетевого устройства отличен от символьного и блочного

Архитектура ввода-вывода



Идентификация символьных драйверов

Для пользовательских процессов символьный драйвер представляется в виде файла устройства в каталоге /dev

Если выполнить команду `ls -l` мы, то символьные драйверы будут помечены с, блочные b

```
crw----- 1 root tty      5,    1 2008-10-30 13:07 console
crw-rw---- 1 root root    10,  252 2008-10-30 16:06 dac960_gam
crw-rw---- 1 root audio   14,    9 2008-10-30 16:06 dmmidi
brw-rw---- 1 root floppy  2,    4 2008-10-30 16:06 fd0d360
brw-rw---- 1 root floppy  2,    8 2008-10-30 16:06 fd0h1200
brw-rw---- 1 root floppy  2,   40 2008-10-30 16:06 fd0h1440
crw-r----- 1 root kmem    1,    2 2008-10-30 16:06 kmem
crw-rw---- 1 root root     1,   11 2008-10-30 16:06 kmsg
crw-rw-r--  1 root lp      6,    0 2008-10-30 13:07 lp0
crw-r----- 1 root kmem    1,    1 2008-10-30 16:06 mem
crw-rw---- 1 root audio   14,    2 2008-10-30 16:06 midi
crw-rw---- 1 root uucp   108,    0 2005-11-21 08:29 ppp
crw-rw-rw-  1 root tty     5,    2 2008-10-30 16:06 ptmx
crw-rw-r--  1 root root     1,    9 2008-10-30 13:07 urandom
crw-rw---- 1 root root    189,    0 2008-10-30 16:06 usbdev1.1
crw-rw---- 1 root root    250,    0 2008-10-30 16:06 usbdev1.1_ep00
crw-rw---- 1 root root    250,    1 2008-10-30 16:06 usbdev1.1_ep81
crw-rw-rw-  1 root root     1,    5 2005-11-21 06:22 zero
```

Драйвер имеет :

major_number -

идентификации
ядром ОС

minor_number -

ядро не
использует,

использует сам
драйвер для того,
чтобы различать
устройства)

User space API для работы с символьным драйвером

```
int open(char *path,int oflags,mode_t permission)
```

```
int close(int fd)
```

```
int read(int fd,char *buff,int count)
```

```
int write(int fd,char *buff,int count)
```

и др.

Пример пользовательского приложения для работы с символьным драйвером

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
#include <fcntl.h>
```

```
#include <stdio.h>
```

```
int main() {
```

```
    int fd;
```

```
    char buf[100];
```

```
    fd = open("/dev/com1", O_RDWR);
```

```
    read(fd, buf, 20);
```

```
    buf[20]=0;
```

```
    printf("Input: >>> %s <<<\n", buf);
```

```
    close(fd);
```

```
}
```

Компоненты символьного драйвера драйвера

- Инициализация (init) инициализация устройства, регистрация драйвера в ядре
- Реализация функций для точек входа (open, close, read, write, ioctl), соответствующих системным вызовам
- Обработчики прерываний, обработчики таймеров, bottom halves, kernel threads

Структуры данных драйвера

1. `struct file_operations`
2. `struct file`

Интерфейс символического драйвера **file_operations** (/linux/fs.h)

struct file_operations {

```
    struct module *owner;                // Pointer to the LKM that owns the structure
    loff_t (*llseek) (struct file *, loff_t, int); // Change current read/write position in a file
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *); // Used to retrieve data from the device
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *); // Used to send data to the device
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t); // Asynchronous read
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t); // Asynchronous write
    ssize_t (*read_iter) (struct kiocb *, struct iov_iter *); // possibly asynchronous read
    ssize_t (*write_iter) (struct kiocb *, struct iov_iter *); // possibly asynchronous write
    int (*iterate) (struct file *, struct dir_context *); // called when VFS needs to read the directory
    contents
    unsigned int (*poll) (struct file *, struct poll_table_struct *); // Does a read or write block?
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long); // Called by the ioctl system call
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long); // Called by the ioctl system call
    int (*mmap) (struct file *, struct vm_area_struct *); // Called by mmap system call
    int (*mremap) (struct file *, struct vm_area_struct *); // Called by memory remap system call
    int (*open) (struct inode *, struct file *); // first operation performed on a device file
    int (*flush) (struct file *, fl_owner_t id); // called when a process closes its copy of the descriptor
    int (*release) (struct inode *, struct file *); // called when a file structure is being released
    int (*fsync) (struct file *, loff_t, loff_t, int datasync); // notify device of change in its FASYNC flag
    int (*aio_fsync) (struct kiocb *, int datasync); // synchronous notify device of change in its FASYNC flag
    int (*fasync) (int, struct file *, int); // asynchronous notify device of change in its FASYNC flag
    int (*lock) (struct file *, int, struct file_lock *); // used to implement file locking
    ...
};
```

Реализация функций драйвером

Для реализации интерфейса драйвер создает структуру `file_operations`.

Символьный драйвер реализует необходимые ему функции.

Для остальных указателей в `file_operations` устанавливается в `NULL`.

```
struct file_operations fops = {  
    .read = device_read,  
    .write = device_write,  
    .open = device_open,  
    .release = device_release  
};
```

инициализация структуры с использованием синтаксиса C99.

Номер устройства

- В ядре для представления пары старший и младший номер устройства используется тип `dev_t`
- Определен в `<linux/kdev_t.h>` в Linux 2.6: 32 bit size (major: 12 bits, minor: 20 bits)
- Макрос для создания номера устройства
`MKDEV(int major, int minor);`
- Макрос для извлечения номера устройства:
`MAJOR(dev_t dev);`
`MINOR(dev_t dev);`

Пример использования

```
int init_module(void)
{
    Major = register_chrdev(0, DEVICE_NAME, &fops);
    if (Major < 0) {
        printk(KERN_ALERT "Registering char device failed with %d\n", Major);
        return Major;
    }
    printk(KERN_INFO "I was assigned major number %d. To talk to\n", Major);
    printk(KERN_INFO "the driver, create a dev file with\n");
    printk(KERN_INFO "'mknod /dev/%s c %d 0'.\n", DEVICE_NAME, Major)

    return SUCCESS;
}
```

Деинициализация драйвера

```
void cleanup_module(void) {  
    /* Удаление символьного драйвера */  
    unregister_chrdev(Major, DEVICE_NAME);  
    printk(KERN_ALERT "Cleanup_module OK \n");  
}
```

Детально file_operation

```
int (*open) ( struct inode *, struct file *);
```

Вызывается, когда открывается файл.

```
int (*release) (struct inode *, struct file *);
```

Вызывается, когда файл закрывается.

Пример

```
static int device_open(struct inode *inode, struct file *file){
    static int counter = 0;
    if (Device_Open)
        return -EBUSY;
    Device_Open++;
    sprintf(msg, "I already told you %d times Hello world!\n", counter++);
    msg_Ptr = msg;
    try_module_get(THIS_MODULE);
    return SUCCESS;
}

static int device_release(struct inode *inode, struct file *file){
    Device_Open--; /* We're now ready for our next caller */
    /** Decrement the usage count, or else once you opened the file, you'll
        never get rid of the module.  */
    module_put(THIS_MODULE);
    return 0;
}
```

Структура file

Структура file, создается когда происходит вызов open
mode_t f_mode;

Режим открытия файла (FMODE_READ and/or FMODE_WRITE)

loff_t f_pos; текущий offset в файле.

struct file_operations *f_op;

struct dentry *f_dentry

Операции записи и чтения

```
ssize_t (*read) (  
    struct file *, /* Open file descriptor */  
    __user char *, /* Userspace buffer to fill up */  
    size_t,        /* Size of the userspace buffer */  
    loff_t *);     /* Offset in the open file */
```

Вызывается при чтении из файла

```
ssize_t (*write) (  
    struct file *, /* Open file descriptor */  
    __user const char *, /* Userspace buffer to write to the device */  
    size_t,          /* Size of the userspace buffer */  
    loff_t *);       /* Offset in the open file */
```

Пример реализации

```
static ssize_t device_read(struct file *filp, char *buffer, /* buffer to fill with data */
                             size_t length, /* length of the buffer */ loff_t * offset)
{
    int bytes_read = 0; /* Number of bytes actually written to the buffer*/
    /* If we're at the end of the message, * return 0 signifying end of file */
    if (*msg_Ptr == 0) return 0;
    /* Actually put the data into the buffer */
    while (length && *msg_Ptr) {
        /** The buffer is in the user data segment, not the kernel
        * segment so "" assignment won't work. We have to use
        * put_user which copies data from the kernel data segment to
        * the user data segment. */
        put_user(* (msg_Ptr++), buffer++);
        length--;
        bytes_read++;
    }
    /* Most read functions return the number of bytes put into the buffer */
    return bytes_read;
}
```

Обмен данными с пользовательским процессом

Драйвер не может использовать memsys для обмена между памятью user space и памятью kernel space

В функции read и write необходимо использовать специальные функции:

```
include <asm/uaccess.h>
```

```
unsigned long copy_to_user (void __user *to, const void *from, unsigned long n);
```

```
unsigned long copy_from_user (void *to, const void __user *from, unsigned long n);
```

Функции в случае успеха возвращают 0

Установка драйвера в ядро

1. Вставить модуль:

```
insmod foodrv.ko
```

2. Посмотреть в `/proc/devices`, какой у драйвера `major_number` (предположим 100)

3. Создать файл устройства:

```
mknod /dev/foodev c 100 0
```

(для автоматического создания файла устройства можно использовать `udev`)

Динамическое выделение памяти

malloc - не существует.

kmalloc/kfree выделяет до 128k физической памяти в последовательном блоке

kcalloc(. . .) аналогично kmalloc, но память обнуляется

```
void *kmalloc (size_t size, int flags);
```

flags определены в <linux/mm.h>

GFP_USER associated userspace process sleeps until free memory available

GFP_KERNEL associated kernel function sleeps until free memory available

GFP_ATOMIC doesn't sleep (used in ISRs)

Литература

1. **The Linux Kernel Module Programming Guide** by P.J. Saltzman, M. Burian, Ori Pomerantz <http://tldp.org/LDP/lkmpg/2.6/html/index.html>
<http://www.opennet.ru/docs/RUS/lkmpg26/> (русский перевод)
2. **The linux-kernel mailing list FAQ** <http://www.tux.org/lkml>.
3. **Rob Day The Kernel Newbie Corner: What's in That Loadable Module, Anyway?** <http://www.linux.com/learn/linux-training/32867-the-kernel-newbie-corner-whats-in-that-loadable-module-anyway>
4. **Andrew Murray Init Call Mechanism in the Linux Kernel**
<http://linuxgazette.net/157/amurray.html>
5. **Бовет.Д. Ядро Linux, 3-е издание.-СПб.: БХВ-Петербург, 2007**
6. **Kernel modules** https://wiki.archlinux.org/index.php/Kernel_modules
(полезные команды и конфигурирование)
7. **Writing a Linux Kernel Module — Part 2: A Character Device**
<http://derekmolloy.ie/writing-a-linux-kernel-module-part-2-a-character-device/>