# Линукс - лаба 7

Список загруженных модулей – `lsmod` (*обращается к файлу /proc/modules*)

Загрузка модуля (пример msdos)

```
insmod /lib/modules/2.6.0/kernel/fs/fat/fat.ko
insmod /lib/modules/2.6.0/kernel/fs/msdos/msdos.ko
```

или просто:

```
modprobe -a msdos
```

## "Hello, World" (часть 1): Простейший модуль ядра.

Создаём файл `cat > hello-1.c`

### Пример 1. hello-1.c

```
/*
 * hello-1.c - Простейший модуль ядра.
 */
#include <linux/module.h> /* Необходим для любого модуля ядра */
#include <linux/kernel.h> /* Здесь находится определение KERN_ALERT */
int init_module(void)
{
 printk("<1>Hello world 1.\n");
 /*
 * Если вернуть ненулевое значение, то это будет воспринято как признак
ошибки,
 * возникшей в процессе работы init_module; в результате модуль не будет
загружен.
 */
 return 0;
}
void cleanup_module(void)
{
 printk(KERN_ALERT "Goodbye world 1.\n");
}
```

`init_module()` вызывает `insmod` во время загрузки модуля, и функцию завершения работы модуля -- `cleanup_module()`, которую вызывает `rmmod`.

## Сборка модулей ядра

Попробуем собрать наш с вами модуль `hello-1.c`

Создаём файл `cat > Makefile`

**Пример 2. Makefile для модуля ядра**

```
obj-m += hello-1.o
```

Для того, чтобы запустить процесс сборки модуля, дайте команду

```
make -C /usr/src/linux-uname -r SUBDIRS=$PWD modules
```

(если у вас в каталоге /usr/src присутствует символическая ссылка linux на каталог с исходными текстами ядра, то команда сборки может быть несколько упрощена:

```
make -C /usr/src/linux SUBDIRS=$PWD modules
```
)

Команда выше почему-то не работает, вот как исправил:

```
sudo make -C /lib/modules/$(uname -r)/build M=$PWD modules
```

А ещё пример 1 нерабочий, в него надо добавить лицензию:

```
MODULE_LICENSE("GPL");
```

Отредачим с помощью `gedit hello-1.c`

После всего на экран должно быть выведено нечто подобное:

```
[root@pcsenonsrv test_module]# make -C /usr/src/linux-`uname -r` SUBDIRS=$PWD
modules
make: Entering directory `/usr/src/linux-2.6.x
 CC [M] /root/test_module/hello-1.o
 Building modules, stage 2.
 MODPOST
 CC /root/test_module/hello-1.mod.o
 LD [M] /root/test_module/hello-1.ko
make: Leaving directory `/usr/src/linux-2.6.x
```

**Обратите внимание:** в ядрах версии 2.6 введено новое соглашение по именованию объектных файлов модулей. Теперь, они имеют расширение .ko (взамен прежнего .o), что отличает их от обычных объектных файлов.

Дополнительную информацию по оформлению Makefile-ов модулей вы найдете в *linux/Documentation/kbuild/makefiles.txt*. Обязательно прочтите этот документ прежде, чем начнете углубляться в изучение Makefile-ов.

Теперь можно загрузить свежесобранный модуль! Дайте команду `insmod ./hello-1.ko` (появляющиеся сообщения о "загрязнении" ядра вы сейчас можете просто игнорировать, вскоре мы обсудим эту проблему).

Любой загруженный модуль ядра заносится в список */proc/modules*. Там можете убедиться, наш модуль стал частью ядра.

Потом выгрузите модуль командой `rmmod hello-1` и загляните в файл */var/log/messages*, здесь вы
увидите сообщения, которые сгенерировал ваш модуль. (*/var/log/kern.log*).

Измените содержимое файла `hello-1.c` так, чтобы функция `init_module()` возвращала бы какое либо ненулевое значение и проверьте -- что получится?

**СПОЙЛЕР:**
тут будет спойлер, когда я сам сделаю

# Hello World (часть 2)

Как мы уже упоминали, начиная с ядра, версии 2.3.13, требования к именованию начальной и конечной функций модуля были сняты. Достигается это с помощью макроопределений `module_init()` и `module_exit()`. Они определены в файле *linux/init.h*. Единственное замечание: начальная и конечная функции должны быть определены выше строк, в которых вызываются эти макросы, в противном случае вы получите ошибку времени компиляции. Ниже приводится пример использования этих макроопределений:

**Пример 3. hello-2.c**

```
/*
 * hello-2.c - Демонстрация использования макроопределений module_init() и
module_exit().
 */
#include <linux/module.h> /* Необходим для любого модуля ядра */
#include <linux/kernel.h> /* Здесь находится определение KERN_ALERT */
```

```
#include <linux/init.h> /* Здесь находятся определения макросов */
static int __init hello_2_init(void)
{
 printk(KERN_ALERT "Hello, world 2\n");
 return 0;
}
static void __exit hello_2_exit(void)
{
 printk(KERN_ALERT "Goodbye, world 2\n");
}
module_init(hello_2_init);
module_exit(hello_2_exit);
```

Добавить сборку второго модуля очень просто:

**Пример 4. Makefile для сборки обоих модулей:**

```
obj-m += hello-1.o
obj-m += hello-2.o
```

Теперь загляните в файл *linux/drivers/char/Makefile*.
Он может рассматриваться как пример полноценного Makefile модуля ядра.
Здесь видно, что ряд модулей жестко "зашиты" в ядро (obj-y), но нигде нет строки obj-m. Почему?
Знакомые с языком сценариев командной оболочки легко найдут ответ.
Все записи вида obj-$(CONFIG_FOO) будут заменены на obj-y или obj-m, в зависимости от значения переменных CONFIG_FOO.
Эти переменные вы сможете найти в файле *.config*, который был создан во время конфигурирования ядра с помощью `make menuconfig` или что-то вроде этого.

# Hello World (часть 3): Макроопределения __init и __exit

Это демонстрация особенностей ядра, появившихся, начиная с версии 2.2.

**Пример 5. hello-3.c**

```
/*
 * hello-3.c - Использование макроопределений __init, __initdata и __exit.
 */
#include <linux/module.h> /* Необходим для любого модуля ядра */
#include <linux/kernel.h> /* Здесь находится определение KERN_ALERT */
#include <linux/init.h> /* Здесь находятся определения макросов */
static int hello3_data __initdata = 3;
```

```
static int __init hello_3_init(void)
{
 printk(KERN_ALERT "Hello, world %d\n", hello3_data);
 return 0;
}
static void __exit hello_3_exit(void)
{
 printk(KERN_ALERT "Goodbye, world 3\n");
}
module_init(hello_3_init);
module_exit(hello_3_exit);
```

## Hello World (часть 4): Вопросы лицензирования и документирования модулей

Если у вас установлено ядро 2.4 или более позднее, то наверняка, во время запуска примеров модулей, вам пришлось столкнуться с сообщениями вида:

```
# insmod hello-3.o
Warning: loading hello-3.o will taint the kernel: no license
 See http://www.tux.org/lkml/#export-tainted for information about tainted
modules
Hello, world 3
Module hello-3 loaded, with warnings
```

В ядра версии 2.4 и выше был добавлен механизм контроля лицензий, чтобы иметь возможность предупреждать пользователя об использовании проприетарного (не свободного) кода. Задать условия лицензирования модуля можно с помощью макроопределения `MODULE_LICENSE()`.

Точно так же, для описания модуля может использоваться макрос `MODULE_DESCRIPTION()`, для установления авторства -- `MODULE_AUTHOR()`, а для описания типов устройств, поддерживаемых модулем -- `MODULE_SUPPORTED_DEVICE()`.

Все эти макроопределения описаны в файле *linux/module.h*. Они не используются ядром и служат лишь для описания модуля, которое может быть просмотрено с помощью `objdump`. Попробуйте с помощью утилиты `grep` посмотреть, как авторы модулей используют эти макросы (в каталоге *linux/drivers*).

**Пример 6. hello-4.c**

```
/*
 * hello-4.c - Демонстрация описания модуля.
 */
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#define DRIVER_AUTHOR "Peter Jay Salzman <p@dirac.org>"
#define DRIVER_DESC "A sample driver"
static int __init init_hello_4(void)
{
 printk(KERN_ALERT "Hello, world 4\n");
 return 0;
}
static void __exit cleanup_hello_4(void)
{
 printk(KERN_ALERT "Goodbye, world 4\n");
}
module_init(init_hello_4);
module_exit(cleanup_hello_4);
/*
 * Вы можете передавать в макросы строки, как это показано ниже:
 */
/*
 * Запретить вывод предупреждения о "загрязнении" ядра, объявив код под GPL.
 */
MODULE_LICENSE("GPL");
/*
 * или определения:
 */
MODULE_AUTHOR(DRIVER_AUTHOR); /* Автор модуля */
MODULE_DESCRIPTION(DRIVER_DESC); /* Назначение модуля */
/*
 * Этот модуль использует устройство /dev/testdevice. В будущих версиях ядра
 * макрос MODULE_SUPPORTED_DEVICE может быть использован
 * для автоматической настройки модуля, но пока
 * он служит исключительно в описательных целях.
 */
MODULE_SUPPORTED_DEVICE("testdevice");
```

# Передача модулю параметров командной строки

Имеется возможность передачи модулю дополнительных параметров командной
строки, но делается это не с помощью `argc/argv`.

### Пример 7. hello-5.c

```
/*
 * hello-5.c - Пример передачи модулю аргументов командной строки.
 */
#include <linux/module.h>
#include <linux/moduleparam.h>
```

```
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/stat.h>
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Peter Jay Salzman");
static short int myshort = 1;
static int myint = 420;
static long int mylong = 9999;
static char *mystring = "blah";
/*
 * module_param(foo, int, 0000)
 * Первый параметр -- имя переменной,
 * Второй -- тип,
 * Последний -- биты прав доступа
 * для того, чтобы выставить в sysfs (если ненулевое значение) на более поздней
стадии.
 */
module_param(myshort, short, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP);
MODULE_PARM_DESC(myshort, "A short integer");
module_param(myint, int, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
MODULE_PARM_DESC(myint, "An integer");
module_param(mylong, long, S_IRUSR);
MODULE_PARM_DESC(mylong, "A long integer");
module_param(mystring, charp, 0000);
MODULE_PARM_DESC(mystring, "A character string");
static int __init hello_5_init(void)
{
 printk(KERN_ALERT "Hello, world 5\n=============\n");
 printk(KERN_ALERT "myshort is a short integer: %hd\n", myshort);
 printk(KERN_ALERT "myint is an integer: %d\n", myint);
 printk(KERN_ALERT "mylong is a long integer: %ld\n", mylong);
 printk(KERN_ALERT "mystring is a string: %s\n", mystring);
 return 0;
}
static void __exit hello_5_exit(void)
{
 printk(KERN_ALERT "Goodbye, world 5\n");
}
module_init(hello_5_init);
module_exit(hello_5_exit);
```

Давайте немножко поэкспериментируем с этим модулем:

```
satan# insmod hello-5.o mystring="bebop" myshort=255
myshort is a short integer: 255
myint is an integer: 420
mylong is a long integer: 9999
mystring is a string: bebop
satan# rmmod hello-5
Goodbye, world 5
satan# insmod hello-5.o mystring="supercalifragilisticexpialidocious" myint=100
myshort is a short integer: 1
myint is an integer: 100
```

```
mylong is a long integer: 9999
mystring is a string: supercalifragilisticexpialidocious
satan# rmmod hello-5
Goodbye, world 5
satan# insmod hello-5.o mylong=hello
hello-5.o: `hello' invalid for parameter mylong
```

# Модули, состоящие из нескольких файлов

Ниже приводится пример модуля, состоящего из двух файлов:

### Пример 8. start.c

```
/*
 * start.c - Пример модуля, исходный текст которого размещен в нескольких
файлах
 */
#include <linux/kernel.h> /* Все-таки мы пишем код ядра! */
#include <linux/module.h> /* Необходим для любого модуля ядра */
int init_module(void)
{
 printk("Hello, world - this is the kernel speaking\n");
 return 0;
}
```

### Пример 9. stop.c

```
/*
 * stop.c - Пример модуля, исходный текст которого размещен в нескольких файлах
 */
#include <linux/kernel.h> /* Все-таки мы пишем код ядра! */
#include <linux/module.h> /* Необходим для любого модуля ядра */
void cleanup_module()
{
 printk("<1>Short is the life of a kernel module\n");
}
```

### Пример 10. Makefile для сборки всех модулей

```
obj-m += hello-1.o
obj-m += hello-2.o
obj-m += hello-3.o
obj-m += hello-4.o
obj-m += hello-5.o
obj-m += startstop.o
startstop-objs := start.o stop.o
```