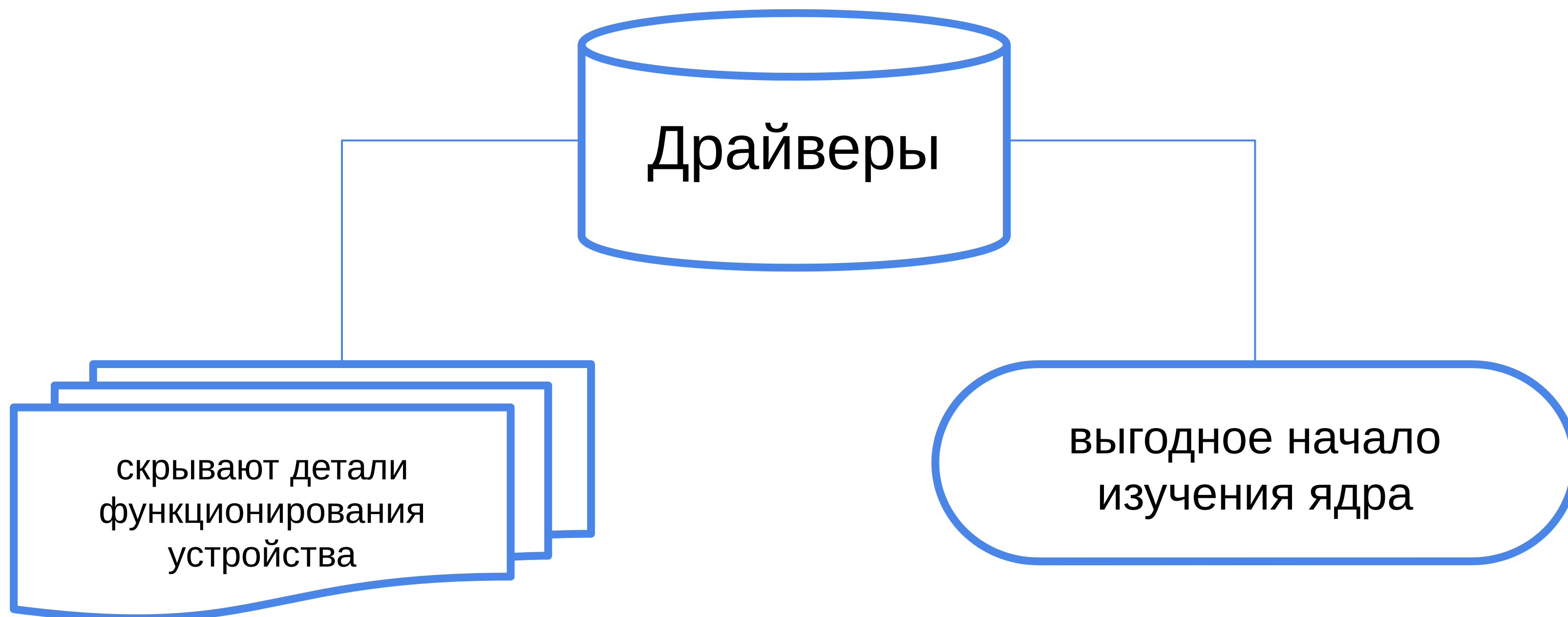
Linux Kernel Training

Kernel modules

Agenda

- Введение в драйверы устройств
- Building module
- Installing Modules
- Generating Module Dependencies
- Загрузка и выгрузка модулей
- Диагностика модуля
- Основные ошибки модуля
- Примечания

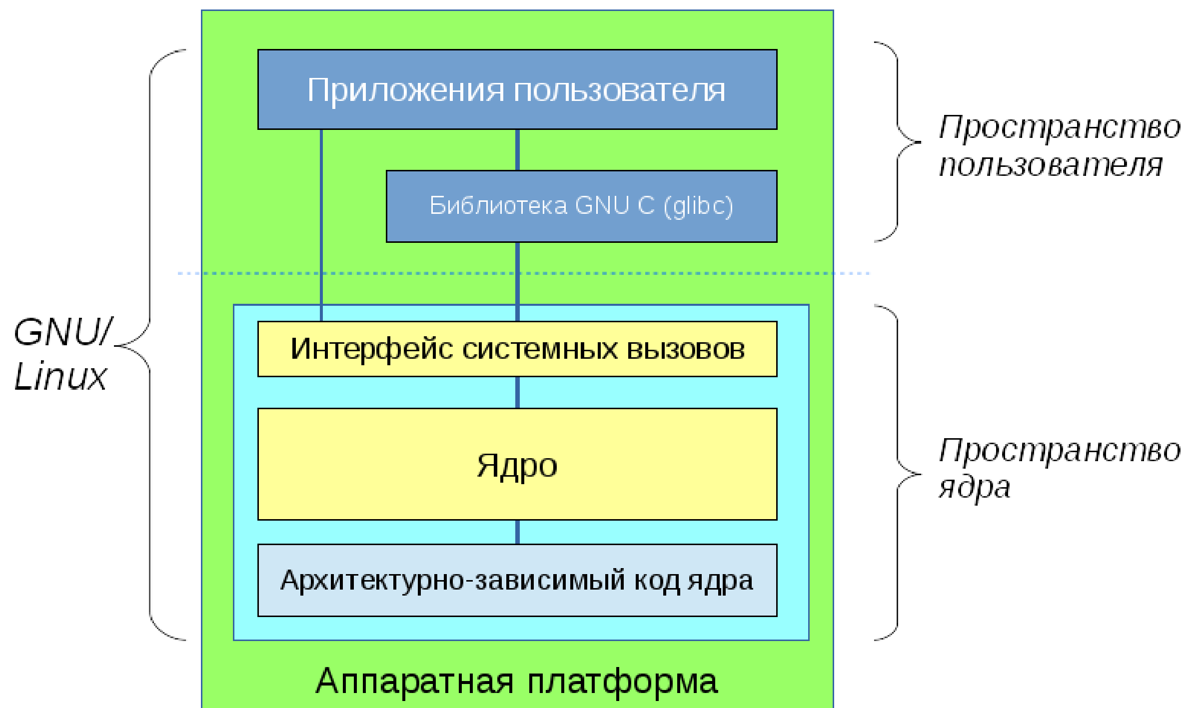

Драйверы
скрывают детали функционирования устройства
выгодное начало изучения ядра

Программный интерфейс

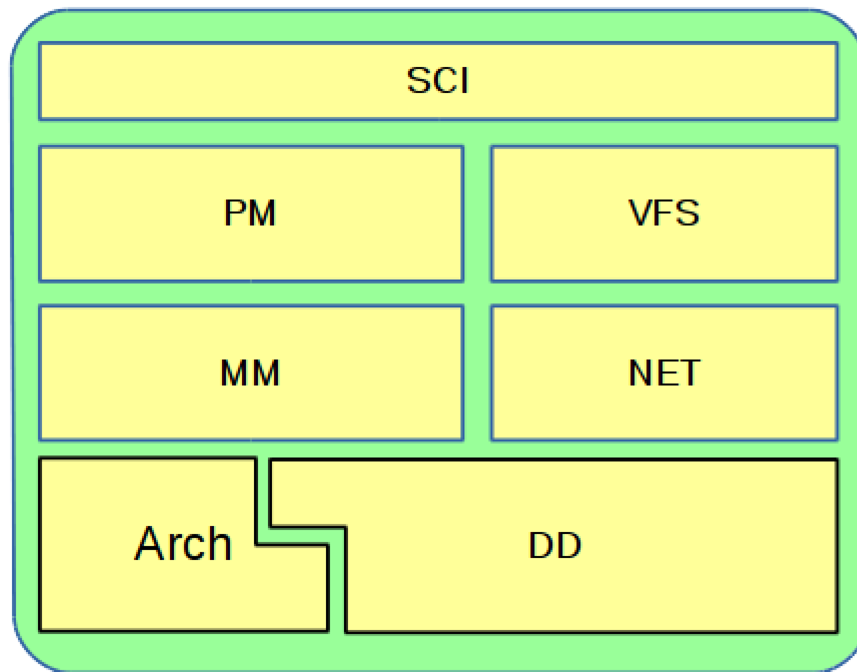
- облегчает написание драйверов в Linux,
- позволяет собирать драйверы отдельно от ядра,
- позволяет подключать их по мере необходимости.

- Преимущество свободных операционных систем - *открытость*
- Драйверы
  - скрывают детали того, как работает устройство
  - хорошее начало изучения ядра
- Программный интерфейс:
  - облегчает написание драйверов в Linux,
  - позволяет собирать драйверы отдельно от ядра,
  - позволяет подключать их по мере необходимости.
- В необходимости написании драйверов устройств для Linux есть достаточно причин, главная из которых — постоянное развитие технологий.

Строение ядра Linux



Основные подсистемы ядра Linux



Управление ресурсами

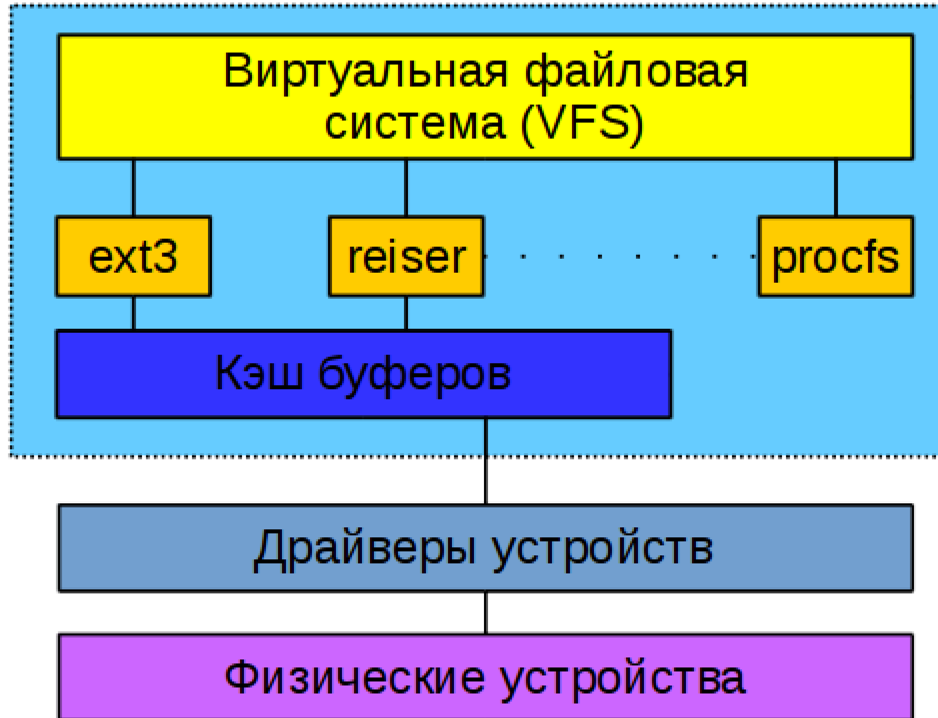
- Управление процессами
  `/linux/kernel`
  `/linux/arch`

- Управление памятью
  `/linux/mm`

- Файловые системы
  `/linux/fs`

- Управление устройствами

Виртуальная файловая система

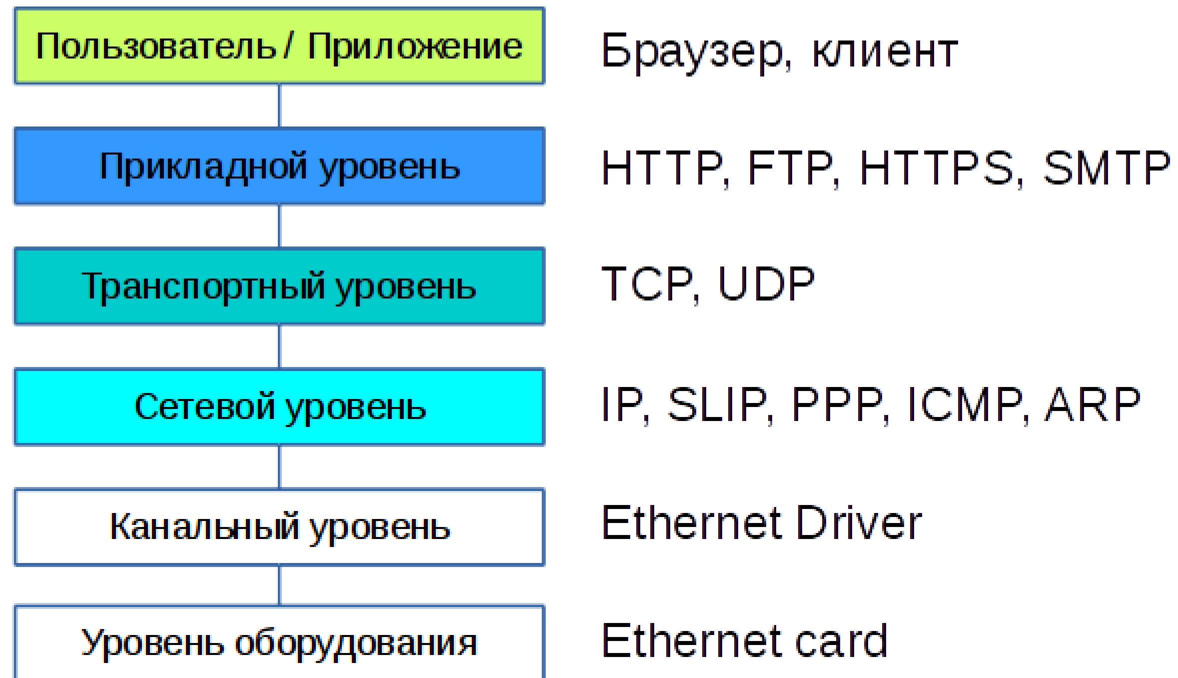


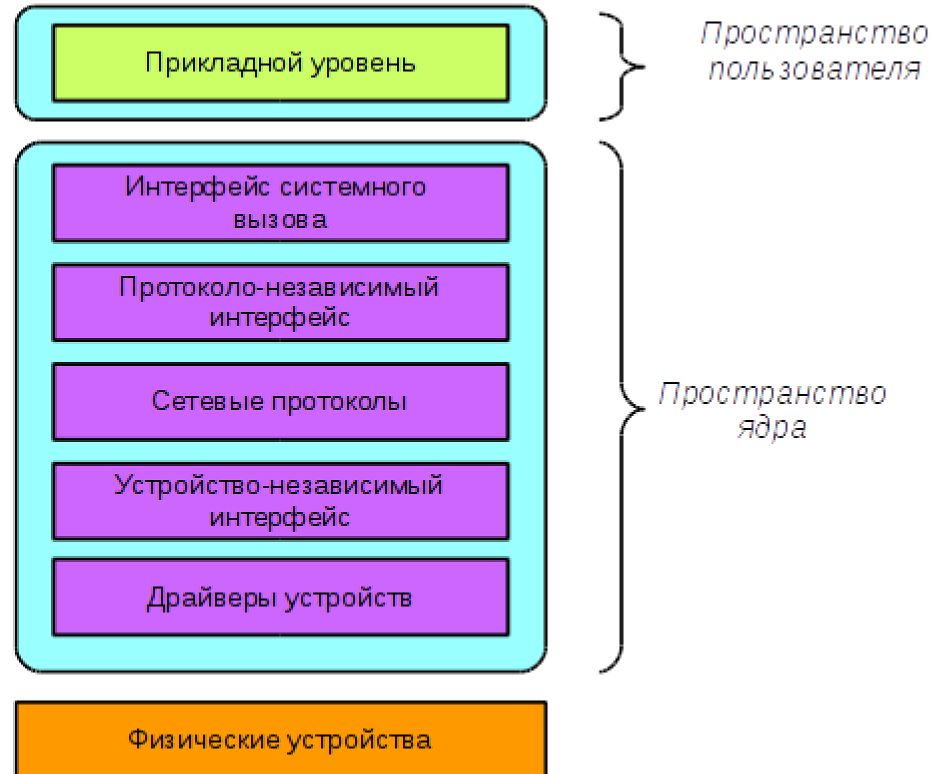
Виртуальная файловая система

Уровень	функции
единая API-абстракция	открытие, закрытие, чтение и запись файлов
абстракции файловых систем	определяют, как реализуются функции верхнего уровня. Они представлены подключаемыми модулями для конкретных файловых систем (которых более 50).
кэш буферов	предоставляет общий набор функций доступа к уровню файловой системы *(независимо от конкретной файловой системы).* оптимизирует доступ к физическим устройствам за счет краткосрочного хранения или упреждающего чтения данных
драйверы устройств	Реализуют интерфейсы для конкретных физических устройств.

Загружаемые модули

Сетевые интерфейсы Linux


Пользователь / Приложение
Браузер, клиент
Прикладной уровень
HTTP, FTP, HTTPS, SMTP
Транспортный уровень
TCP, UDP
Сетевой уровень
IP, SLIP, PPP, ICMP, ARP
Канальный уровень
Ethernet Driver
Уровень оборудования
Ethernet card

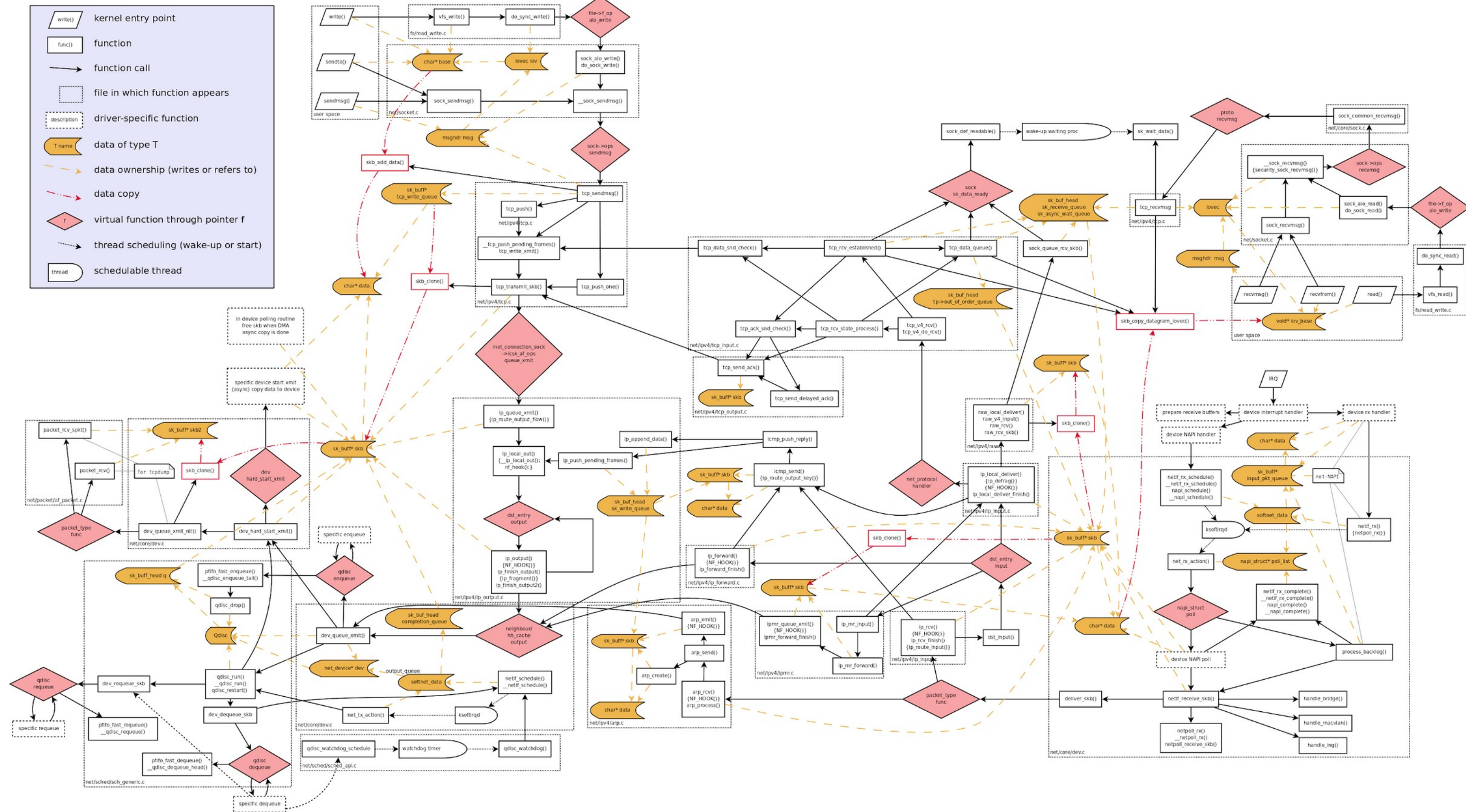

Прикладной уровень
Интерфейс системного вызова
Протоколо-независимый интерфейс
Сетевые протоколы
Устройство-независимый интерфейс
Драйверы устройств
Физические устройства
Пространство пользователя
Пространство ядра

Сетевые соединения (TCP) - поточно-ориентированные

Сетевые устройства обычно разработаны для передачи и приема пакетов.

Сетевой драйвер об отдельных соединениях не знает - он только обрабатывает **пакеты**

Сетевой интерфейс не поточно-ориентированное устройство



kernel entry point
function
function call
file in which function appears
driver-specific function
data of type T
data ownership (writes or refers to)
data copy
virtual function through pointer f
thread scheduling (wake-up or start)
schedulable thread
write()
vfs_write()
do_sync_write()
sendto()
sendmsg()
user space
fs/read_write.c
net/socket.c
sock_sendmsg()
__sock_sendmsg()
tcp_sendmsg()
tcp_push()
tcp_transmit_skb()
tcp_push_one()
net/ipv4/tcp.c
ip_queue_xmit()
ip_local_out()
ip_push_pending_frames()
ip_append_data()
icmp_push_reply()
icmp_send()
dst_entry output
net/ipv4/ip_output.c
dev_queue_xmit()
dev_hard_start_xmit()
dev_queue_xmit_nit()
net/core/dev.c
packet_rcv_spkt()
packet_rcv()
net/packet/af_packet.c
qdisc_run()
dev_requeue_skb
dev_dequeue_skb
pfifo_fast_enqueue()
pfifo_fast_requeue()
pfifo_fast_dequeue()
qdisc_drop()
net/sched/sch_generic.c
qdisc_watchdog_schedule
watchdog timer
qdisc_watchdog()
net/sched/sch_api.c
net_tx_action()
ksoftirqd
arp_xmit()
arp_send()
arp_create()
arp_rcv()
net/ipv4/arp.c
ip_forward()
net/ipv4/ip_forward.c
ipmr_queue_xmit()
ip_mr_input()
ip_mr_forward()
net/ipv4/ipmr.c
ip_rcv()
dst_input()
net/ipv4/ip_input.c
ip_local_deliver()
raw_local_deliver()
net/ipv4/raw.c
tcp_v4_rcv()
tcp_rcv_state_process()
tcp_rcv_established()
tcp_data_queue()
tcp_data_snd_check()
tcp_ack_snd_check()
net/ipv4/tcp_input.c
tcp_send_ack()
tcp_send_delayed_ack()
net/ipv4/tcp_output.c
sock_def_readable()
wake-up waiting proc
sk_wait_data()
tcp_recvmsg
skb_copy_datagram_iovec()
sock_common_recvmsg()
net/core/sock.c
sock_recvmsg()
net/socket.c
recvmsg()
recvfrom()
read()
vfs_read()
do_sync_read()
IRQ
device interrupt handler
device rx handler
prepare receive buffers
device NAPI handler
netif_rx()
net_rx_action()
process_backlog()
netif_receive_skb()
deliver_skb()
handle_bridge()
handle_macvlan()
handle_ing()
device NAPI poll

другие классификации модулей драйверов

- модули USB,
- последовательные модули,
- модули SCSI,
- и так далее.

USB устройство

- символьное устройство (USB-serial),
- блочное устройство (USB-cardreader),
- сетевой интерфейс (USB-network device)).

Загружаемые модули

- позволяют расширение функциональности ядра без остановки
- позволяют вводить в систему при необходимости,
  - во время загрузки
  - по желанию пользователя
- подключены в работающее ядро с помощью **`insmod`**
- отключены **`rmmod`**

Building module

Для сборки модуля ядра необходим **Makefile**

```
CURRENT = $(shell uname -r)
KDIR = /lib/modules/$(CURRENT)/build
PWD = $(shell pwd)
DEST = /lib/modules/$(CURRENT)/misc

TARGET = hello_printk
obj-m := $(TARGET).o

$(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
@rm -f *.o .*.cmd .*.flags *.mod.c *.order
@rm -f .*.*.cmd *.symvers *~ *.*~ TODO.*
@rm -fR .tmp*
@rm -rf .tmp_versions
```

Модуль создан.

```
$ ls *.ko
hello_printk.ko
```

Формат модуля - объектный ELF формат, дополнительными именами:

```
__mod_author5,
__mod_license4,
__mod_srcversion23,
__module_depends,
__mod_vermagic5
```

они определяются специальными модульными макросами.

Installing Modules

Расположение скомпилированных модулей **/lib/modules/{version}/kernel/**

```
/lib/modules/${version}/kernel/drivers/char/fishing.ko
```

устанавливаем скомпилированные модули в нужное место:

```
# make modules_install
```

```
$ modinfo ./hello_printk.ko
```

Загружаем модуль и исследуем его выполнение командами:

```
$ sudo insmod ./hello_printk.ko

$ lsmod | head -n2

$ sudo rmmod hello_printk

$ lsmod | head -n2

$ dmesg | tail -n2

$ sudo cat /var/log/messages | tail -n3
```

Generating Module Dependencies

Утилиты модуля ядра Linux различают зависимости.

Информация о зависимости должна быть ***сгенерирована***.

Чтобы создать информацию о зависимости модуля, необходимо выполнить команду

 # depmod

Для быстрого обновления, перестраивая только информацию для новых модулей:

 # depmod – A

Информация о зависимостях модулей хранится в файле

/lib/modules/${version}/modules.dep

Загрузка и выгрузка модулей

Для загрузки модуля используется команда insmod.

 # insmod module.ko

 # insmod fishing.ko

Для выгрузки модуля используется команда rmmod.

 # rmmod *module*

Для загрузки модуля в ядро необходимо выполнить команду:

 # modprobe *module* **[module parameters]**

Команда **modprobe** также может использоваться для удаления модулей из ядра:

 # modprobe -r *modules*

modprobe -r также удаляет все модули, от которых зависит данный модуль, если они не используются.

lsmod - выдаёт список модулей, загруженных в данный момент

/proc/modules и /sys/module/

modinfo *modulename* — выдает информацию о модуле ядра

Внутренний формат модуля

Собранный модуль является **объектным** файлом ELF формата:

```
$ file hello_printk.ko
```

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

Анализ объектных файлов производим с помощью утилиты **objdump**

```
$ objdump -h hello_printk.ko
```

```
hello_printk.ko: file format elf32-i386
Sections:
Idx Name Size VMA LMA File off Algn
...
1 .text      00000000 00000000 00000000 00000058 2**2
             CONTENTS, ALLOC, LOAD, READONLY, CODE
2 .exit.text 00000015 00000000 00000000 00000058 2**0
             CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
3 .init.text 00000011 00000000 00000000 0000006d 2**0
             CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
...
5 .modinfo   0000009b 00000000 00000000 000000a0 2**2
             CONTENTS, ALLOC, LOAD, READONLY, DATA
6 .data      00000000 00000000 00000000 0000013c 2**2
             CONTENTS, ALLOC, LOAD, DATA
...
8 .bss 00000000 00000000 00000000 000002a4 2**2
             ALLOC
...
```

Список имен модуля

```
$ objdump -t hello_printk.ko
hello_printk.ko: file format elf32-i386
SYMBOL TABLE:
...
00000000 l F .exit.text 00000015 hello_exit
00000000 l F .init.text 00000011 hello_init
00000000 l O .modinfo 00000026 __mod_author5
00000028 l O .modinfo 0000000c __mod_license4
...
```

Еще один способ анализа объектного файла:

```
$ readelf -s hello_printk.ko

Symbol table '.symtab' contains 35 entries:
Num: Value Size Type Bind Vis Ndx Name
...
22: 00000000 21 FUNC LOCAL DEFAULT 3 hello_exit
23: 00000000 17 FUNC LOCAL DEFAULT 5 hello_init
24: 00000000 38 OBJECT LOCAL DEFAULT 8 __mod_author5
25: 00000028 12 OBJECT LOCAL DEFAULT 8 __mod_license4
...
```

Отличия формата

модуля *.ko от обыкновенного объектного формата *.o

```
$ ls -l *.o *.ko
$ modinfo hello_printk.o
$ modinfo hello_printk.ko
```

В *.ko добавлены внешние имена
(srcversion, depends, vermagic),

их значения используются системой для контроля возможности корректной загрузки модуля.

Диагностика модуля

printk() --> syslogd

/lib/modules/`uname -r`/build/include/linux/kernel.h

https://elixir.bootlin.com/linux/v2.6.37.3/source/kernel/printk.c#L622 :

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

const - константа квалификатор уровня сообщений

https://elixir.bootlin.com/linux/v2.6.37.3/source/include/linux/printk.h

```
#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           */
```

через procfs:

```
$ cat /proc/sys/kernel/printk
3       4       1       7
^
+-- максимальный уровень сообщений,
    выводимых в текстовую консоль = 3


# echo 8 > /proc/sys/kernel/printk

$ cat /proc/sys/kernel/printk
8       4       1       7
```

Основные ошибки модуля

```
insmod: can't read './params': No such file or directory
insmod: error inserting './params.ko': -1 Operation not permitted
insmod: error inserting './params.ko': -1 Invalid module format
insmod: error inserting './params.ko': -1 File exists
insmod: error inserting './params.ko': -1 Invalid parameters
insmod: ERROR: could not insert module ./test.ko: Unknown symbol in
module
rmmod: ERROR: Removing 'params': Device or resource busy
```

Примечания

1. printk(), вместо printf()

API для user space стандартизированы (POSIX и др.) и хорошо документированы;

API kernel space могут СУЩЕСТВЕННО изменяться от одной версии ядра к другой

2. # insmod ./hello_printk.ko а не : # insmod hello_printk.ko

При установке модуля: # insmod ./hello_printk.ko (имя файла)

Но при выгрузке: # rmmod hello_printk (имя модуля в RAM)

Let’s Go!