

Мы с вами частично знакомы с некоторым функционалом ядра – оно отвечает за time sharing, управление процессами, их приоритетами и т.п, также управление памятью – та же виртуальная память, swar и всё что с этим связано, ну и из недавнего – отвечает за проверку прав на файлы - каким пользователям к каким файлам есть доступ. Это далеко не всё, чем занимается ядро, что-то мы ещё будем разбирать по мере изучения, но пока давайте разберём, что из себя представляет ядро и что администратору с ним делать.

```
user@centos8:~  
[user@centos8 ~]$ ls /boot/vmlinuz-*  
/boot/vmlinuz-0-rescue-91222dde95634287a2336070235dc625  
/boot/vmlinuz-4.18.0-147.8.1.el8_1.x86_64  
/boot/vmlinuz-4.18.0-147.el8.x86_64  
[user@centos8 ~]$ uname -r  
4.18.0-147.8.1.el8_1.x86_64  
[user@centos8 ~]$ du -h /boot/vmlinuz-*  
7.8M    /boot/vmlinuz-0-rescue-91222dde95634287a2336070235dc625  
7.8M    /boot/vmlinuz-4.18.0-147.8.1.el8_1.x86_64  
7.8M    /boot/vmlinuz-4.18.0-147.el8.x86_64  
[user@centos8 ~]$
```

Для начала, ядро - это программа. В отличие от других программ, оно лежит в директории /boot и называется vmlinuz - ls /boot. Почему оно лежит здесь и что это за другие файлы – это касается вопроса загрузки операционной системы, что мы будем разбирать в другой раз. Как вы видите, тут несколько файлов с названием vmlinuz и они отличаются версиями. Когда мы обновили систему, у нас появилась новая версия ядра, но старая не удалась – если с новым ядром будут проблемы, всегда можно загрузиться со старого. Версию ядра, которую мы сейчас используем, можно увидеть с помощью команды uname -r. Давайте посмотрим, сколько же весит ядро: для этого воспользуемся утилитой du, которая показывает размеры файлов, с ключом -h – чтобы отображалось не в килобайтах, а в более удобном для чтения виде - du -h /boot/vmlinuz-*. Как видите, ядро весит почти 8 мегабайт. На самом деле, буква z в слове vmlinuz говорит о том, что ядро сжато. То есть, фактически, оно весит чуть больше.

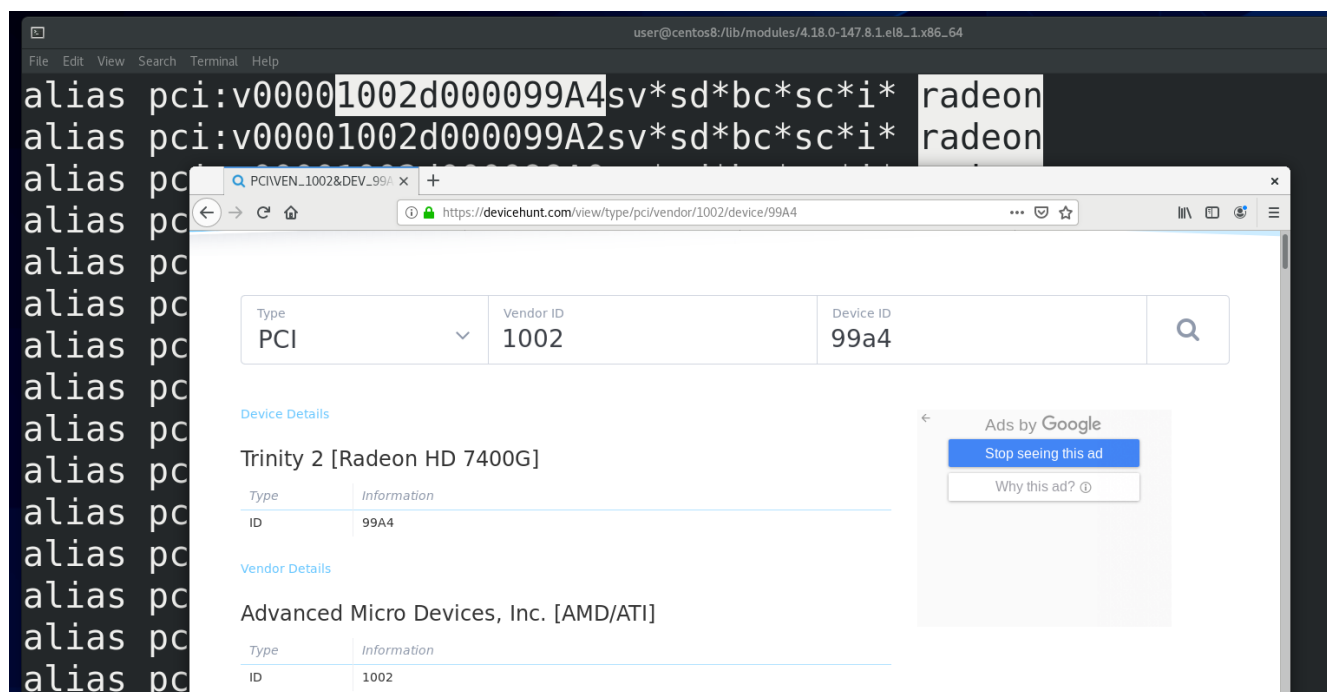
Возможно, вы знаете – ядро Linux используется везде: android смартфоны, коих больше 3 миллиардов, работают на Linux; огромное количество сетевого оборудования, серверов, всяких медиабоксов, умных телевизоров, холодильников, машин, да даже бортовые компьютеры Space X – всё это работает на Linux. Это огромное количество разнообразного оборудования, которое должно поддерживать ядро. Поэтому в разработке ядра участвуют тысячи крупнейших компаний и специалистов. И всё ради 8 мегабайтного файла? Конечно нет. В этом файле только основной функционал, необходимый для работы – работа с памятью, управление процессами и т.п. Когда же ядру нужен дополнительный функционал – допустим, чтобы работать с сетевым адаптером, видеокарты или другим оборудованием – ядро обращается к специальным файлам, называемым модулями. В модулях хранится код, необходимый для работы с оборудованием или программный функционал – допустим, драйвер для видеокарты или программа для шифрования. Обычно это происходит незаметно для пользователя – вы вставили флешку, а ядро загрузило модуль для работы с usb флешками, а также модуль для работы с файловой системой на этой флешке. На других операционных системах это может работать по другому – есть различные архитектуры ядер операционных систем. У Linux архитектура модульная – то есть какой-то

функционал вынесен в модули и подгружается по необходимости. Также Linux называют монолитным – потому что всё что делает ядро происходит в рамках одной программы – а правильнее сказать – все части ядра работают в одном адресном пространстве. Помните, мы обсуждали, что это такое, когда говорили о процессах? Но так как у Linux-а огромный функционал, который бессмысленно держать одновременно в памяти – поэтому функционал вынесен в модули, благодаря чему ускоряется работа ядра.

```
user@centos8:/lib/modules/4.18.0-147.8.1.el8_1.x86_64
File Edit View Search Terminal Help
[user@centos8 ~]$ ls /lib/modules
4.18.0-147.8.1.el8_1.x86_64 4.18.0-147.el8.x86_64
[user@centos8 ~]$ cd /lib/modules/$(uname -r)
[user@centos8 4.18.0-147.8.1.el8_1.x86_64]$ ls
bls.conf          modules.alias.bin  modules.drm        source
build             modules.block      modules.modetesting symvers.gz
config            modules.builtin    modules.networking System.map
extra             modules.builtin.bin modules.order       updates
kernel            modules.dep         modules.softdep     vdso
misc              modules.dep.bin     modules.symbols     vmlinux
modules.alias     modules.devname     modules.symbols.bin weak-updates
[user@centos8 4.18.0-147.8.1.el8_1.x86_64]$ less modules.alias
```

Так вот, модули ядра хранятся в директории /lib/modules/ - ls /lib/modules, где есть директории для каждой версии установленного ядра. Зайдём в директорию текущего ядра - cd /lib/modules/\$(uname -r); ls и посмотрим файл modules.alias - less modules.alias. Тут перечислено - для каких устройств какие модули грузить в ядро.

В отличие от Windows, где вы ставите систему, а потом доустанавливаются драйвера, в Linux большинство драйверов уже предустановлены в виде модулей. Это благодаря тому, что многие производители оборудования сотрудничают с разработчиками ядра и предоставляют открытый исходный код драйверов на оборудование. Но, естественно, далеко не все производители предоставляют исходный код своих драйверов, из-за чего что-то может не работать из коробки – зачастую, это касается драйверов на wi-fi. Иногда, допустим, в случае с драйверами на видеокарты Nvidia, находятся энтузиасты, которые с помощью реверс инжиниринга создают свободные драйвера – т.е. берут драйвер с закрытым исходным кодом, изучают его с помощью специальных программ и методик и стараются воссоздать этот драйвер. Для видеокарт Nvidia таким образом создан драйвер nouveau. Зачастую это работает – естественно без каких-либо гарантий, потому что драйвер написан не самим производителем. При этом сам производитель – тот же Nvidia, также предоставляет свой драйвер в виде модуля, но уже с закрытым исходным кодом, т.е. проприетарный, поэтому он не бывает включён в ядро по умолчанию, из-за чего нужно самому доустановить этот модуль. К примеру, после установки Centos на Virtualbox, мы с вами установили гостевые дополнения Virtualbox вручную именно потому, что они не под лицензией GPL, хотя сам Virtualbox имеет открытый исходный код с лицензией GPL. Всё это к тому, что если вы поставили Linux и у вас что-то не работает, допустим, wifi, то, скорее всего, производитель wifi адаптера не открыл исходный код своих драйверов и вам придётся искать нужный драйвер на сайте производителя, либо гуглить. Однако, некоторые юзер-френдли дистрибутивы, допустим Ubuntu, делают это за вас – после того, как вы установите Ubuntu, система найдёт нужные проприетарные драйвера и предложит вам их установить, что удобно для новичков.



Возвращаясь к нашему файлу, во втором столбике у нас перечислены идентификаторы оборудования – так называемые hardware id. Возьмём для примера radeon - /radeon – это видеокарты от AMD. Чтобы было удобнее, откроем сайт devicehunt.com – где мы можем увидеть информацию о вендорах и оборудовании. Так вот, если ядро видит, что к pci шине подключено устройство, у которого vendor id – 1002, а device id – 99A4 – ядро знает, что для этого устройства нужен модуль с названием radeon. Тут могут быть ещё версии какого-то оборудования, классы и подклассы – но нам это сейчас не особо важно. Если вам интересно, что значат все эти обозначения, посмотрите по [ссылке](#).

```
user@centos8:/lib/modules/4.18.0-147.8.1.el8_1.x86_64
filename:      /lib/modules/4.18.0-147.8.1.el8_1.x86_64/kernel/drivers/gp
u/drm/radeon/radeon.ko.xz
license:      GPL and additional rights
description:   ATI Radeon
author:       Gareth Hughes, Keith Whitwell, others.
:
alias:        pci:v00001002d0000131Dsv*sd*bc*sc*i*
alias:        pci:v00001002d0000131Csv*sd*bc*sc*i*
alias:        pci:v00001002d0000131Bsv*sd*bc*sc*i*
:
parm:         tv:TV enable (0 = disable) (int)
parm:         audio:Audio enable (-1 = auto, 0 = disable, 1 = enable) (i
nt)
parm:         disp_priority:Display Priority (0 = auto, 1 = normal, 2 =
:

```

Сам этот файл не статичный, он генерируется от информации из самих модулей. Чтобы увидеть информацию о каком-нибудь модуле, можно использовать команду `modinfo` – допустим, `modinfo radeon - modinfo radeon | less`. Тут мы видим расположение и имя файла, причём, у всех модулей расширение `.ko`, ну а `.xz` в конце означает, что модуль в сжатом виде. Также лицензия, автор, описание. И чуть ниже у нас `alias`-ы – собственно на основе этого и генерируется файл `modules.alias`, который мы смотрели. Ну и ниже – параметры – функционал оборудования на уровне драйвера. Допустим, `audio` – будет ли у нас видекарта работать со звуком через тот же `hdmi`.

```
user@centos8:~
[user@centos8 ~]$ ls /sys/
block  class  devices  fs          kernel  power
bus    dev    firmware hypervisor  module
[user@centos8 ~]$ ls /sys/bus/pci/devices/00*
'/sys/bus/pci/devices/0000:00:00.0':
ari_enabled          dma_mask_bits      modalias            revision
broken_parity_status driver_override     msi_bus             subsystem
class                enable              numa_node           subsystem_device
config               firmware_node       power               subsystem_vendor
consistent_dma_mask_bits irq                  remove              uevent
d3cold_allowed       local_cpulist       rescan              vendor
device               local_cpus          resource

```

Так вот, недавно мы узнали о виртуальной файловой системе `procfs`, через которую ядро нам показывает информацию о процессах. А для структурированной информации об устройствах и драйверах есть виртуальная файловая система `sysfs`, доступная в директории `/sys` `ls /sys;` `ls /sys/bus/pci/device/00*`. И хотя тут куча файлов, через которые можно увидеть очень много информации, сидеть и копаться в этих файлах не всегда удобно, есть утилиты, которые показывают эту же информацию в более компактном и простом виде.

```
user@centos8:~  
File Edit View Search Terminal Help  
[user@centos8 ~]$ lscpu  
Architecture:          x86_64  
CPU op-mode(s):        32-bit, 64-bit  
Byte Order:             Little Endian  
CPU(s):                 1  
  
user@centos8:~  
File Edit View Search Terminal Help  
[user@centos8 ~]$ sudo lshw  
[sudo] password for user:  
centos8  
      description: Computer  
      product: VirtualBox  
      vendor: innotek GmbH  
      version: 1.2
```

Например, `lscpu` – показывает информацию о процессоре, `lspci` – показывает информацию об устройствах, подключённых на pci шину, `lsusb` – устройства, подключённые к usb. Для более подробной информации вы можете использовать `lshw - sudo lshw`, а из графических утилит есть `hardinfo`.

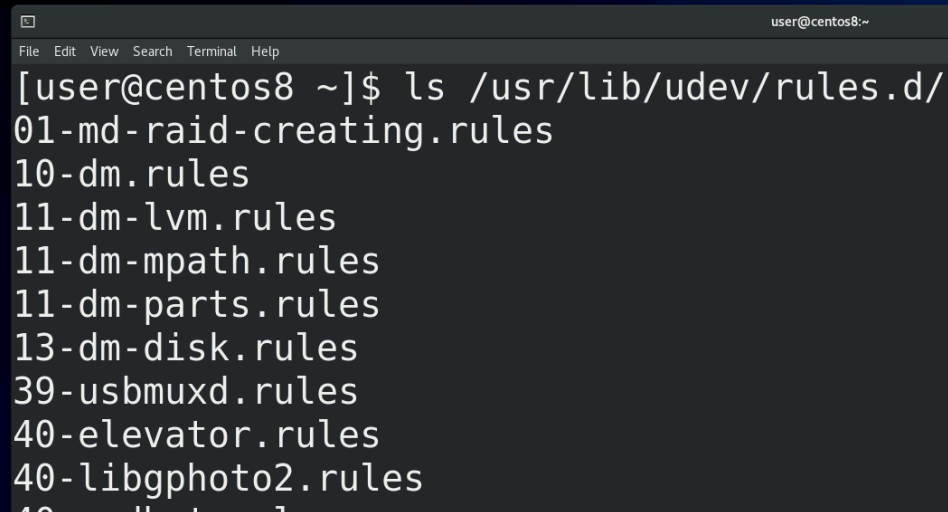
```
user@centos8:~  
File Edit View Search Terminal Help  
trol: RX  
[ +0.001168] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not ready  
[ +0.052074] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not ready  
[ +0.705776] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready  
[ +2.484575] virbr0: port 1(virbr0-nic) entered blocking state  
[ +0.000236] virbr0: port 1(virbr0-nic) entered listening state  
[ +0.070235] virbr0: port 1(virbr0-nic) entered disabled state  
[ +3.465436] 11:15:49.685090 main      VBoxService 6.1.16 r140961 (verbosi  
ty: 0) linux.amd64 (Oct 15 2020 16:40:53) release log  
      11:15:49.685091 main      Log opened 2021-01-15T11:15:49.6850  
8400007
```

Чтобы видеть, что происходит в ядре при запуске системы или сейчас, например, вы вставили флешку и хотите понять, видит ли её система или нет, вы можете использовать утилиту `dmesg - sudo dmesg -wH`. Запустили команду, вставили флешку или любое другое устройство, и тут вы увидите, как ядро распознаёт устройство.

Ядро, при виде какого-нибудь оборудования или при необходимости работы с каким-нибудь программным функционалом, загружает модули автоматически. И хотя работать с этим вам придётся не так часто, вы должны иметь представление, как это работает и как это менять. Например, может быть требование, чтобы система игнорировала флешки, хотя, по умолчанию, вы вставили флешку и она работает.

```
[user@centos8 ~]$ sudo modprobe radeon
[sudo] password for user:
[user@centos8 ~]$ lsmod | grep radeon
radeon                1626112  0
i2c_algo_bit          16384    1 radeon
drm_kms_helper        217088    3 vmwgfx,vboxvideo,radeon
ttm                   110592    3 vmwgfx,vboxvideo,radeon
drm                   524288    9 vmwgfx,drm_kms_helper,vboxvideo,radeon,ttm
[user@centos8 ~]$ sudo modprobe -r radeon
[user@centos8 ~]$ sudo modprobe -r vboxguest
modprobe: FATAL: Module vboxguest is in use.
```

И так, для управления модулями ядра используется утилита modprobe. Допустим, если хотим загрузить модуль radeon, пишем - `sudo modprobe radeon`. Увидеть можем с помощью утилиты lsmod, которая показывает загруженные модули - `lsmod | grep radeon`. Одни модули могут работать с другими и сами могут использоваться какими-то процессами. Например, чтобы выгрузить модуль из ядра, можно использовать тот же modprobe с ключом -r - `sudo modprobe -r radeon`. Этот модуль не использовался, поэтому мне получилось с лёгкостью его выгрузить. Но есть модуль, допустим, vboxguest, который используется и убрать его с помощью modprobe не получится - `sudo modprobe -r vboxguest`. Предварительно нужно избавиться от процессов, которые используют этот модуль. Например, чтобы выгрузить драйвер видеокарты, вам нужно будет предварительно завершить все приложения, использующие графический интерфейс.

A terminal window titled 'user@centos8:~' with a menu bar (File, Edit, View, Search, Terminal, Help). The command `ls /usr/lib/udev/rules.d/` has been executed, listing several rule files in the directory.

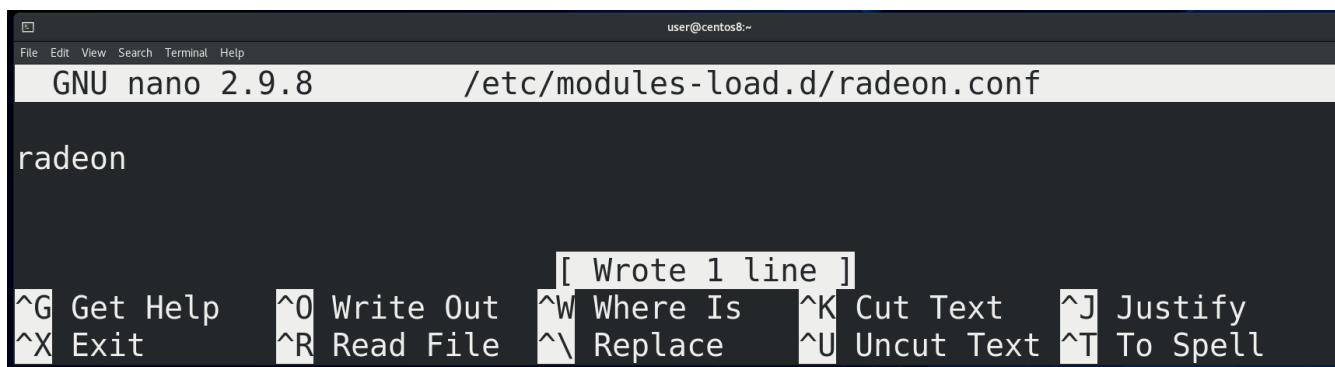
```
[user@centos8 ~]$ ls /usr/lib/udev/rules.d/
01-md-raid-creating.rules
10-dm.rules
11-dm-lvm.rules
11-dm-mpath.rules
11-dm-parts.rules
13-dm-disk.rules
39-usbmuxd.rules
40-elevator.rules
40-libgphoto2.rules
40-netdata.rules
```

Так вот, возвращаясь к теме про автоматическую загрузку модулей. В системе есть программа, называемая udev – именно она отвечает за управление устройствами. И когда ядро видит новое устройство, оно создаёт событие, которое отслеживает udev. У udev есть большое количество правил - `ls /usr/lib/udev/rules.d/`, которые оно применяет при виде того или иного события. Например, udev видит в событии, что в usb подключено устройство, которое говорит

что оно является накопителем – и у udev есть правило, которое при виде такого устройства создаёт для него специальный файл в директории /dev с таким-то названием – допустим, sdb. Этот файл ядро связывает с драйвером, работающим с оборудованием. Таким образом наша флешка становится файлом, благодаря чему мы через этот файл можем взаимодействовать с устройством. Точно также можно в udev прописать, чтобы при виде usb устройства для него передавался специальный параметр, который бы запрещал устройству что-либо делать. Углубляться в udev мы пока не будем, для начала хватит понимания, зачем он вообще нужен.

```
[user@centos8 ~]$ cat /lib/modules/$(uname -r)/modules.builtin
kernel/arch/x86/crypto/glue_helper.ko
kernel/arch/x86/crypto/aes-x86_64.ko
kernel/arch/x86/crypto/aesni-intel.ko
kernel/arch/x86/crypto/sha1-ssse3.ko
kernel/arch/x86/crypto/sha256-ssse3.ko
kernel/arch/x86/kernel/msr.ko
kernel/arch/x86/kernel/cpuid.ko
kernel/arch/x86/platform/intel/iosf_mbi.ko
kernel/mm/zpool.ko
kernel/mm/zbud.ko
kernel/mm/zsmalloc.ko
```

Модули можно скомпилировать в ядро – тогда отпадает необходимость загружать модули из файлов, но это увеличивает размер ядра. Такие модули называются встроенными и их список можно увидеть в файле modules.builtin - cat /lib/modules/\$(uname -r)/modules.builtin. А те модули, которые загружаются при необходимости и которые можно, теоретически, выгрузить, называются загружаемыми. Встроенные же модули выгрузить из ядра не получится.



```
user@centos8~
File Edit View Search Terminal Help
GNU nano 2.9.8 /etc/modules-load.d/radeon.conf

radeon

[ Wrote 1 line ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text ^T To Spell
```

Иногда может понадобиться, чтобы загружаемые модули загружались всегда при включении компьютера. Для этого нужно создать файл в директории /etc/modules-load.d/ с .conf в конце имени, допустим, radeon.conf - sudo nano /etc/modules-load.d/radeon.conf, где вписываем название модуля, которые мы хотели бы загружать в ядро при включении. Теперь, после перезагрузки, этот модуль автоматом загрузится.

```
# User changes in this file are preserved across upgrades.
#
# For Intel
#options kvm_intel nested=1
#
# For AMD
#options kvm_amd nested=1
```

Если же нам нужно, чтобы какой-то определённый модуль не загружался при включении или загружался с определёнными параметрами – теми параметрами, которые мы видели с `modinfo radeon` - то для этого нужно создать файл в директории `/etc/modprobe.d` тоже заканчивающийся на `.conf`, например, `kvm.conf` - `sudo nano /etc/modprobe.d/kvm.conf`. Тут у нас есть закомментированные примеры. Чуть подробнее об этом можно почитать на [арчвики](#).

Надеюсь у вас сложилось представление, что же есть ядро. Это далеко не весь функционал ядра, я его и не смогу полностью рассмотреть. Но теперь вы немного знаете про само ядро, про его модули, про драйвера и устройства, в том числе `udev`, который позволит вам более гибко настраивать работу системы с устройствами и директорию `dev`, где у нас специальные файлы, с помощью которых мы можем работать с различным оборудованием.