

# Операционные системы

## лекция 2

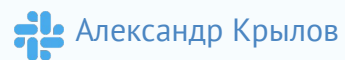


Александр  
Крылов



## **Александр Крылов**

Lead DevOps services в ПАО СК Росгосстрах



---


# План модуля

1. Работа в терминале, лекция 1
2. Работа в терминале, лекция 2
3. Операционные системы, лекция 1
4. **Операционные системы, лекция 2**
5. Файловые системы
6. Компьютерные сети, лекция 1
7. Компьютерные сети, лекция 2
8. Компьютерные сети, лекция 3
9. Элементы безопасности информационных систем



# План занятия

1. [Оценка потребления ресурсов](#)
2. [Ядро, модули](#)
3. [Системы инициализации: systemd](#)
4. [Итоги](#)
5. [Домашнее задание](#)



# Оценка потребления ресурсов

# free

```
root@netology1:~# free -m
```

	total	used	free	shared	buff/cache
Mem:	981	175	143	0	661
Swap:	979	4	975		

Сколько памяти на самом деле доступно в системе?

# free

```
root@netology1:~# free -m
```

	total	used	free	shared	buff/cache
Mem:	981	175	<b>143</b>	0	<b>661</b>
Swap:	979	4	975		

**143** - неправильный ответ, т.к. он показывает вообще ничем не занятую память, buff/cache включают в себя в частности страничный кэш блочных устройств, который ускоряет доступ к горячим данным, но может быть в любой момент сброшен:

```
root@netology1:~# echo 3 >/proc/sys/vm/drop_caches
root@netology1:~# free -m
```

	total	used	free	shared	buff/cache
Mem:	981	178	<b>696</b>	0	<b>106</b>
Swap:	979	4	975		

Часть данных, которая оказалась не сброшена, – буферы, т.е. временно занятая для операций ядром или приложениями память.

# Типичные времена доступа

execute typical instruction	$1/1,000,000,000 \text{ sec} = 1 \text{ nanosec}$
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec



# top, atop, htop, mpstat

- **top** – посмотреть список активных процессов в системе с сортировкой по потребляемому процессорному времени (по умолчанию), Shift + F - для интерактивного меню, где s – меняет сортировку и d – включает/выключает отображение колонки.
- **atop** – программа для записи исторического top с заданным интервалом. Возможно, с учетом современных подходов к сбору метрик и eBPF уже не столь актуальна, но может быть полезна. t / Shift + t для перехода по интервалам, atop -r <log> -b <time> для старта с нужного времени.

*echo LOGINTERVAL=10 > /etc/default/atop; systemctl restart atop*

- **htop** – более современный вариант top. top был написан задолго до эпохи многоядерных систем, htop представляет нагрузку с учетом визуализации отдельных ядер.
- **mpstat** – аналогичная по сути тестовая утилита с более подробной статистикой.

```
~ # mpstat -P ALL 1
Linux 4.19.102+ (api-6f74c84dfb-wlrrz) _x86_64_ (12 CPU)
18:53:40      CPU      %usr    %nice    %sys  %iowait    %idle
18:53:41     all      9.55     0.00     2.09     0.00     88.11
18:53:41        0      1.00     0.00     0.00     0.00     97.00
18:53:41        5     26.73     0.00     6.93     0.00     65.35
18:53:41        9     30.00     0.00     7.00     0.00     63.00
18:53:41       10     33.33     0.00     6.06     0.00     60.61
```

# iostat, iftop, nethogs, sar -n DEV 1

- **iostat** – аналогичная top утилита, только для дисковой подсистемы. Далеко не всегда, даже на специализированном сервере (баз данных) очевидно, какой процесс вызывает высокую загрузку дисков, iostat поможет в этом разобраться.
- Само по себе знание о загрузке сетевого линка не говорит нам о том, на что действительно расходуется полоса. **iftop** поможет в этом.
- iftop показывает сетевое взаимодействие без привязки к процессом. Для подобной статистики запустите **nethogs <interface>**.
- Нередко на хосте бывает несколько сетевых интерфейсов, и может быть важно оценить загрузку по ним. В такой ситуации поможет **sar -n DEV 1**.

wget <https://mirror.yandex.ru/ubuntu/dists/focal/main/installer-amd64/current/legacy-images/netboot/boot.img.gz>

# node\_exporter

Приведенные выше инструменты хороши для **интерактивной** оценки производительности системы, но они не решают проблемы сбора **исторических** данных.

Решение: актуальная связка Prometheus + [node\\_exporter](#).

С коллектором вы познакомитесь в модуле по мониторингу, но на node\_exporter логично взглянуть на этой лекции.

```
vagrant@netology1:~/node_exporter-1.0.1.linux-amd64$ curl -s localhost:9100/metrics  
2>/dev/null | grep node_filesystem_avail_bytes | grep mapper  
node_filesystem_avail_bytes{device="/dev/mapper/vgvgagrant-root",fstype="ext4",mountpoint="/" } 6.075752448e+10
```

```
vagrant@netology1:~/node_exporter-1.0.1.linux-amd64$ df -h | grep mapper  
/dev/mapper/vgvgagrant-root 62G 1.6G 57G 3% /
```



# Ядро, модули

# Ядро и дистрибутив

**Посмотреть версию ядра, с которой работает ОС:**

```
root@netology1:~# uname -r
5.4.0-31-generic
```

**Посмотреть дистрибутив:**

```
root@netology1:~# cat /etc/issue
Ubuntu 20.04 LTS \n \l
root@netology1:~# lsb_release -a
No LSB modules are available.
Description:    Ubuntu 20.04 LTS
```

**Посмотреть, с какой конфигурацией ядро собрано:**

```
root@netology1:~# grep -v ^# /boot/config-$(uname -r) | tail -n2
CONFIG_PUNIT_ATOM_DEBUG=m
CONFIG_UNWINDER_FRAME_POINTER=y
```

**В современных ОС ядра распространяются в форме пакетов, как и другие приложения:**

```
root@netology1:~# dpkg -l | grep linux-image-5
ii  linux-image-5.4.0-31-generic  5.4.0-31.35   amd64      Signed kernel image generic
```

Установочные скрипты сами обновляют записи загрузчика, initrd и vmlinuz (бинарный образ ядра).

# Модули ядра

По формальной классификации ядро Linux – монолитное. Несмотря на это, **есть возможность динамически подгружать и выгружать модули.**

Посмотреть загруженные:

```
vagrant@netology1:~$ lsmod | wc -l  
87
```

Классическая **команда для загрузки нового модуля – insmod** (insert module), однако есть более удобный современный вариант – **modprobe**, который умеет автоматически подгружать зависимости и не требует указания пути:

```
vagrant@netology1:~$ modinfo virtio_net | grep depends  
depends:          net_failover  
... sudo insmod /lib/modules/$(uname -r)/kernel/drivers/net/virtio_net.ko  
insmod: ERROR: could not insert module  
/lib/modules/5.4../kernel/drivers/net/virtio_net.ko: Unknown symbol in module  
  
vagrant@netology1:~$ sudo modprobe virtio_net
```

# Настройка ядра – параметры загрузки

В части параметров настройки ядро Linux аналогично обычным приложениям - оно принимает строку параметров на вход.

```
vagrant@netology1:~$ cat /proc/cmdline  
BOOT_IMAGE=/boot/vmlinuz-5.4.0-31-generic root=/dev/mapper/vg_vagrant-root  
ro net.ifnames=0 biosdevname=0 quiet
```

В данном примере:

- загрузчик передал путь до образа, из которого ядро загружено;
- на каком устройстве находится корневая файловая система;
- монтировать корневую файловую систему в RO режиме во время загрузки;
- параметры для подсистемы udev об именовании сетевых интерфейсов;
- “тихая” загрузка без сообщений.

# Настройка ядра – параметры загрузки

Настройки загрузчика специфичны для дистрибутива.

В Ubuntu изменить их можно через приведенный файл и запустив **sudo update-grub**.

```
vagrant@netology1:~$ grep GRUB_CMDLINE_LINUX /etc/default/grub
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
GRUB_CMDLINE_LINUX="net.ifnames=0 biosdevname=0 "
```

Если доступа в загруженную ОС нет, параметры ядра можно поменять во время работы загрузчика через его сервисное меню (**init=/bin/sh** или **single** для single user mode).



# Настройка ядра – sysctl

Параметров ядра существуют больше тысячи - передавать их в командной строке и каждый раз перезагружаться неудобно. Для решения данной проблемы существует механизм **sysctl**.

```
root@netology1:~# sysctl -a | grep 'v4.ip_forward '  
net.ipv4.ip_forward = 0  
root@netology1:~# sysctl -w net.ipv4.ip_forward=1  
net.ipv4.ip_forward = 1  
root@netology1:~# sysctl net.ipv4.ip_forward  
net.ipv4.ip_forward = 1
```

Внесенные таким образом изменения не сохраняются после перезагрузки. Чтобы сделать их персистентными, присутствует каталог **/etc/sysctl.d**. В дистрибутивах обычно ряд настроек заданы отличными от стандартных, поэтому ознакомьтесь с имеющимися там параметрами до внесения своих.

```
root@netology1:~# grep -v '^#' /etc/sysctl.d/* | grep = | column -t | head -n2  
/etc/sysctl.d/10-console-messages.conf:kernel.printk           = 4      4  1  7  
/etc/sysctl.d/10-ipv6-privacy.conf:net.ipv6.conf.all.use_tempaddr = 2
```

Применить изменения (без указания **-p** загрузит **/etc/sysctl.conf**):

```
root@netology1:~# sysctl -p /etc/sysctl.d/10-ipv6-privacy.conf  
net.ipv6.conf.all.use_tempaddr = 1  
net.ipv6.conf.default.use_tempaddr = 2
```

# dmesg и syslog

Два важных источника информации об ОС:

```
root@netology1:~# dmesg -T | tail -n2
[19:27:11 2020] e1000: eth1 NIC Link is Up 1000 Mbps Full Duplex...
[19:27:11 2020] IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready
```

**1. dmesg** – инструмент доступа к записям ядра. В **dmesg** (ключ -T – конвертировать время в человеко-читаемое) присутствует информация о загрузке ОС, об изменении состояния аппаратных ресурсов. *Проблемы с дисками, отвалившиеся сетевые карты, ошибки при коррекции ошибок ECC – многие обращающие на себя внимание сообщения найдут свое место здесь.*

**2. syslog** – штатный логгер, куда попадают сообщения уже от приложений в юзер-спейсе. Мест, куда приложения могут писать логи, – множество: от локальных файлов до сервисов вроде journald. Однако **/var/log/syslog** остается местом, куда стоит заглянуть при оценке состояния системы.



# Системы инициализации: `systemd`

# init – PID 1, процессы ядра

Мы узнали, что новые процессы в Linux создаются клонированием родительского процесса. Из этого правила есть понятное исключение – первый процесс в системе, который ядро создает самостоятельно и назначает ему PID 1, – init.

**init** – собирательное название для контроллера инициализации ОС, а не конкретная технология.

**Система инициализации отвечает за:**

- достижение корректного состояния на разных этапах загрузки;
- запуска служб самой ОС и прикладных программ в нужном порядке;
- монтирование файловых систем;
- обратный процесс при выключении ОС;
- init должен уметь “усыновлять” процессы, ставшие “сиротами”.

init – не единственный процесс, который создан ядром. Процессы в **ps** в квадратных скобках – “ядерные” процессы, что видно по 0 размеру RSS памяти в юзер-спейсе.

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	2	0.0	0.0	0	0	?	S	Jul06	0:00	[kthreadd]
root	20	0.0	0.0	0	0	?	S	Jul06	0:00	\_ [oom_reaper]

# systemd, systemctl

Систем инициализации было разработано много, но стандарт для большинства популярных дистрибутивов сегодня – systemd.

systemd многие критикуют за несоответствие философии Unix, согласно которой у программы должно быть единственное предназначение, которое она должна выполнять хорошо.

systemd же, будучи системой инициализации, сегодня заменяет множественные сервисы ОС: логирование, планировщик задач cron, даже локальный кеширующий DNS и синхронизация времени есть в стандартной поставке, что вообще никак не связано с системой инициализации.

Базовый команды:

- `systemctl list-units --all` (посмотреть все *юниты* под управлением systemd)
- `systemctl status nginx.service` (посмотреть статус работы *сервиса*)  
сразу же виден путь расположения *юнит-файла*, который отвечает за сервис (`/lib/systemd/system/nginx.service` в нашем случае)
- `systemctl start/stop/restart/reload nginx`  
(`.service` можно опустить если типа юнита один)
- `systemctl enable/disable nginx` (автозапуск сервиса включен/выключен)

# journalctl, systemctl

- `systemctl cat nginx.service`
- `systemctl edit --full nginx.service`
- `systemctl daemon-reload` (перечитать измененные юнит-файлы самого systemd)
- `systemctl list-dependencies nginx.service` (посмотреть зависимости сервиса nginx)
- `journalctl -b -u nginx.service` (посмотреть логи сервиса nginx, которые были записаны с момента загрузки, b от boot)

# Неймспейсы процессов

Вы слышали про контейнеры – **Docker, LXC** и пр. Все базовые технологии для контейнеризации предоставляет само ядро Linux, а известные названия – инструменты, которые делают обращение с ними удобнее.

**Основная концепция контейнеров – пространства имен.** Все, о чем мы с вами говорили, будь то командный интерпретатор `bash`, процессы-потомки, запущенные из него, – все это тоже находилось в пространстве имен по умолчанию.

```
root@netology1:~# lsns
```

NS	TYPE	NPROCS	PID	USER	COMMAND
4026531835	<b>cgroup</b>	112	1	root	/sbin/init
4026531836	<b>pid</b>	112	1	root	/sbin/init
4026531837	<b>user</b>	112	1	root	/sbin/init
4026531838	<b>uts</b>	110	1	root	/sbin/init
4026531839	<b>ipc</b>	112	1	root	/sbin/init
4026531860	<b>mnt</b>	1	15	root	kdevtmpfs
4026531992	<b>net</b>	112	1	root	/sbin/init

**Вызов `lsns`** покажет, что **типов неймспейсов присутствует несколько**. Сейчас нас интересует пространство имен `PID`. В модуле про виртуализацию/контейнеризацию мы рассмотрим подробнее, как запустить полноценный контейнер только лишь средствами ядра, но и сейчас можно попробовать выполнить элементы этого касательно пространства имен `PID`.

# Системный вызов unshare

Таким нехитрым образом можно организовать полностью изолированный от хостового неймспейс, в котором PID 1 будет иметь команда, которую мы запустили.

```
root@netology1:~# screen
root@netology1:~# unshare -f --pid --mount-proc /bin/bash
root@netology1:~# ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.3	9836	3980	pts/0	S	07:29	0:00	/bin/bash
root	8	0.0	0.3	11476	3316	pts/0	R+	07:29	0:00	ps aux

Из хостового NS при этом наш процесс имеет совершенно обычный PID:

```
root      237934  0.0  0.0  8080   596 pts/3  S   07:30  0:00 \_ unshare -f --pid..
root      237935  0.0  0.3  9836  4012 pts/3  S+  07:30  0:00 \_ /bin/bash
```

Имея привилегии, можно “зайти” в этот namespace с хоста:

```
root@netology1:~# nsenter --target 237935 --pid --mount
root@netology1:/# ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.3	9836	4012	pts/3	S+	07:30	0:00	/bin/bash
root	28	0.0	0.4	9836	4168	pts/0	S	07:38	0:00	-bash
root	37	0.0	0.3	11476	3396	pts/0	R+	07:38	0:00	ps aux



# Дополнительные материалы

Сайт и книги [Brendan Gregg](#), в частности:

- [USE-метод](#)
- [Широкий список утилит для мониторинга](#)

Пример быстрого интерактивного анализа хоста на практике:

- [Netflix](#)

---

## Итоги

- Рассмотрели некоторые утилиты для оценки ресурсов системы.
- Познакомились с systemd.
- Затронули неймспейсы ядра на примере ns PID.



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и  
пишите отзыв о лекции!**

**Александр Крылов**