

ИУ-10
Системное
Программное
Обеспечение
Администрирование Linux
Процессы.
Systemd и его возможности.

На этом уроке

1. Узнаем, как происходит процесс загрузки ОС.
2. Выясним, что такое процесс и как управлять процессами.
3. Познакомимся с главным процессом в системе - systemd
4. Разберём простейшие утилиты мониторинга процессов.

Содержание

На этом уроке	1
Процесс	2
Атрибуты процессов	2
Сигналы	3
Systemd	4
Файлы Модулей/Юнитов	6
Service.Unit	7
Target.Unit	8
Управление опциями юнитов	10
Создание юнита	11
Мониторинг процессов	12
Практическое задание	16
Глоссарий	16
Дополнительные материалы	17
Используемые источники	17

Процесс

Для всего, что происходит на сервере Linux, запускается процесс. Процесс - это совокупность кода, выполняющегося в памяти компьютера. Есть приложения, которые могут создавать в результате своей работы не один, а несколько процессов. Когда процесс запущен, он может использовать несколько потоков (thread). Поток - это задача, запускаемая процессом, которую может обслуживать выделенное ядро процессора.

Можно выделить три основных типа процессов:

- Пользовательские процессы (shell job) - это команды, запускаемые из командной строки. Они связаны с оболочкой, которая была текущей на момент запуска процесса.
- Демоны - это процессы, предоставляющие сервисы. Обычно они запускаются при загрузке системы и часто (но, конечно, не во всех случаях) запускаются с правами root.
- Потоки ядра являются частью ядра Linux. Вы не можете управлять ими с помощью обычных инструментов, но для мониторинга производительности системы важно за ними следить.

Процесс может находиться в разных состояниях. Вот некоторые из них:

1. Процесс работает — состояние когда код процесса выполняется.
2. Процесс «спит» — состояние, когда процесс ожидает каких-то событий (ввода данных с клавиатуры и т. п.).
3. Процесс остановлен — процесс был остановлен, чаще всего при помощи сигналов.
4. Процесс «зомби» — процесс уже не существует, его код и данные уже выгружены из оперативной памяти компьютера, но запись в таблице процессов сохранилась. **Примечание!** Подобное состояние процессов крайне не желательно.

Атрибуты процессов

У каждого процесса есть набор важных атрибутов:

1. **PID и PPID** — идентификатор процесса (Process Identifier) и идентификатор родительского процесса (Parent Process Identifier). Числовое значение, присваиваемое каждому процессу: от 1 до 65535. Процессы

в ОС Linux имеют древовидную структуру. При старте запускается процесс `systemd` с номером 1, который порождает все остальные процессы. Процесс получает при запуске свои **PID** и **PPID**. PID процесса всегда уникален, но при этом у нескольких процессов может быть одинаковый **PPID**. Операционная система взаимодействует с процессами через **PID**.

2. **UID** — владелец процесса, пользователь, от которого запущен процесс.
3. **TTY** - терминал, из которого запущен процесс
4. **TIME** - общее время процессора, затраченное на выполнение процессора
5. **CMD** — команда, запустившая процесс.

Список запущенных процессов и указанных атрибутов позволит увидеть команда **ps -ef**.

```
root@server:~# ps -ef | grep -P
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	655	1	0	Jan08	?	00:00:00	sshd: /usr/sbin/sshd -D [listener] 0 of 10-100
root	5413	655	0	08:54	?	00:00:00	sshd: user

Сигналы

Пользовательские процессы, как правило, живут до окончания пользовательской сессии, но бывают случаи, когда какое-то приложение или запущенный процесс необходимо завершить аварийно (оно банально зависло). Такие ситуации возникают не только с пользовательскими процессами, но и с процессами-демонами. Для завершения работы существуют специальные сигналы, которые мы можем передать процессу, используя команду **kill**. Полный список сигналов можно получить, выполнив команду **kill -l**.

Из 28 сигналов стоит выделить следующие три:

- Сигнал **SIGTERM (15)** используется для запроса остановки процесса.
- Сигнал **SIGKILL (9)** используется для принудительной остановки процесса.

- Сигнал **SIGHUP** (1) используется для приостановки процесса. В результате процесс перечитывает свои файлы конфигурации, что делает этот сигнал полезным для внесения изменений в файл конфигурации процесса.

Команда **kill** работает с процессом через его **PID** или **PPID**. `kill -15 <PID>` посылает процессу сигнал **SIGTERM**, который обычно заставляет процесс прекращать свою деятельность и в спокойном режиме закрывать все открытые файлы.

Сигнал **SIGTERM** работает не всегда, потому что процессы имеют право его игнорировать (условия игнорирования определяются разработчиками ПО). В этом случае используется `kill -9 <PID>`, чтобы послать процессу сигнал **SIGKILL**. Поскольку сигнал **SIGKILL** нельзя игнорировать, он заставляет процесс останавливаться в любой ситуации, даже с риском потери данных. Помимо этого, передав сигнал на принудительное завершение **PPID** (ID родительского процесса) `kill -9 <PPID>`, мы завершим всё дерево процессов, порождённых этим процессом, что может сильно дестабилизировать систему.

```
root@server:~# ps -ef | grep nginx
root          702          1  0 16:05 ?                00:00:00 nginx: master
/usr/sbin/nginx -g daemon on; master_process on;
www-data      703          702  0 16:05 ?                00:00:00 nginx: worker
root         1408        987  0 17:38 pts/0          00:00:00 grep --color=auto nginx
root@server:~# kill -9 702

root@server:~# ps -ef | grep nginx
root         1410        987  0 17:39 pts/0          00:00:00 grep --color=auto nginx
root@server:~#
```

В этом примере мы нашли PID процесса - 702, у которого есть дочерний процесс - 703. Передав `kill -9 702`, мы убили сразу оба процесса.

Systemd

Systemd - это самый масштабный процесс, который управляет запуском оборудования, монтированием файловых систем, стартом служб, ограничением потребления ресурсов и многим другим. По сути это менеджер

всего в User Space.

Важным в работе `systemd` является понятие юнита. Юнит или модуль - это абстрактный элемент, попадающий под управление `systemd`. Юнитом могут быть службы, точки монтирования, сервисы, работающие по времени, и многое другое.

Полный перечень юнитов можно получить с помощью команды `systemctl -t help`.

```
root@server:~# systemctl -t help
Available unit types:
service
mount
swap
socket
target
device
automoun
t timer
path
slice
scope
```

Systemctl — основная команда для управления и мониторинга **systemd**, позволяет получать информацию о состоянии системы и запущенных службах, а также управлять службами.

Основные параметры **systemctl**:

1. `systemctl status` выведет на экран состояние системы.
2. `systemctl` выведет список запущенных юнитов.
3. `systemctl [start|stop|status|restart|reload] service_name` позволит запустить службу (start), остановить (stop), получить информацию о службе (status), перезапустить службу (restart), перечитать конфигурационный файл службы (reload).

```
root@server:~# systemctl status nginx.service
• nginx.service - A high performance web server and a reverse proxy server
  Loaded: loaded (/lib/systemd/system/nginx.service; disabled; vendor
  preset: enabled)
  Active: active (running) since Wed 2021-01-06 21:16:21 UTC; 2 days ago
    Docs: man:nginx(8)
  Main PID: 659 (nginx)
    Tasks: 2 (limit: 2282)
  Memory: 11.6M
  CGroup: /system.slice/nginx.service
```

```
└─659 nginx: master process /usr/sbin/nginx -g daemon on;
master_process on;
    └─661 nginx: worker process

Jan 06 21:16:20 server systemd[1]: Starting A high performance web server
and a reverse proxy server...
Jan 06 21:16:21 server systemd[1]: Started A high performance web server
and a reverse proxy server.
```

Примечание! *Служба `nginx.service` является примером юнита `systemd`.* Из вывода можно получить информацию о состоянии процесса, времени работы, используемой памяти, небольшую часть логов.

4. `systemctl [enable|disable] service_name` позволит добавить (enable) или убрать (disable) службу из автозагрузки.

Файлы Модулей/Юнитов

Основное преимущество работы с Systemd по сравнению с предыдущими методами состоит в том, что он предоставляет единый интерфейс для управления юнитами. Этот интерфейс определяется файлами конфигурации модулей, которые могут находиться в трех местах:

- **`/usr/lib/systemd/system`** содержит файлы модулей по умолчанию, которые были установлены из репозитория с пакетами. Вы никогда не должны редактировать эти файлы напрямую.
- **`/etc/systemd/system`** содержит файлы настраиваемых модулей. Он также может содержать файлы, которые были написаны администратором или сгенерированы командой редактирования `systemctl`.
- **`/run/systemd/system`** содержит файлы модулей, которые были созданы автоматически.

Если файл юнита существует более чем в одном из этих расположений, модули в каталоге **`/run/systemd/system`** имеют наивысший приоритет и переписывают любые настройки, определенные где-либо еще. Юниты в **`/etc/systemd/system`** имеют второй по приоритету, а юниты в **`/usr/lib/systemd/system`** идут последними.

Описанные файлы являются стандартными текстовыми файлами. Давайте посмотрим на их содержимое.

Service.Unit

Наиболее важным типом юнита является **service**, так как он используется для запуска процессов, служб. Рассмотрим пример конфигурации **sshd.service**.

```
root@server:~# systemctl cat sshd.service
# /lib/systemd/system/ssh.service
[Unit]
Description=OpenBSD Secure Shell server
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target auditd.service
ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Service]
EnvironmentFile=-/etc/default/ssh
ExecStartPre=/usr/sbin/sshd -t
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS
ExecReload=/usr/sbin/sshd -t
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartPreventExitStatus=255
Type=notify
RuntimeDirectory=sshd
RuntimeDirectoryMode=0755

[Install]
WantedBy=multi-user.target
Alias=sshd.service
```

- **[Unit]** описывает модуль и определяет зависимости. Важным является параметр **After**, указывающий, после чего следует сервис запускать. В нашем случае служба удаленного доступа ssh должна запускаться после юнитов `network.target` и `auditd.service`. Иногда в секции может присутствовать опция **Before**, которая указывает, что этот модуль должен быть запущен раньше определенных юнитов.
- **[Service]** описывает, как запускать (**ExecStart**), останавливать службу (**ExecStop**), выполнять считывание конфигурационных файлов (**Reload**). Обратите внимание на параметр `Type`, который используется для указания способа запуска процесса. Параметр **Restart** со значением `on-failure` указывает, что процесс необходимо автоматически перезапустить в случае нежелательной остановки, падения.
- **[Install]** указывает, в каком `target` необходимо запустить данный юнит. `Targets` будут рассмотрены дальше.

Target.Unit

Target.Unit является особым типом юнита, используемого для выбора очередности загрузки модулей. Target unit - это по сути группа юнитов, объединенных по какому-то общему признаку или необходимых для старта определенных функций сервера. К примеру **printer.target** используется для запуска или остановки всех устройств, необходимых для обеспечения функциональности печати.

Сами по себе target могут зависеть между собой. Вы можете использовать команду `systemctl list-dependencies` для обзора любых существующих зависимостей. **Basic.target** определяет все юниты, которые всегда должны запускаться.

```
root@server:~# systemctl cat basic.target | grep -vP '^\s*#'

[Unit]
Description=Basic System
Documentation=man:systemd.special(7)
Requires=sysinit.target
Wants=sockets.target timers.target paths.target slices.target
After=sysinit.target sockets.target paths.target slices.target tmp.mount

RequiresMountsFor=/var /var/tmp
Wants=tmp.mount
```

Requires определяет юниты, которые загружаются вместе с модулем читаемого файла. Если один из перечисленных юнитов будет деактивирован, этот юнит также будет деактивирован.

Wants определяет, какие юниты необходимо попытаться (не обязательно с положительным результатом) загрузить до старта просматриваемого модуля. При использовании команды `systemctl enable nginx` изменяется состав этой директивы при помощи изменения содержимого каталога `/etc/systemd/system`. В этом каталоге можно найти подкаталоги вида ***.target.***, содержащий символические ссылки на службы, которые должны быть запущены и попадающие под параметр **Wants**.

```
root@server:~# systemctl enable nginx
Synchronizing state of nginx.service with SysV service script with
/lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable nginx
Created symlink /etc/systemd/system/multi-user.target.wants/nginx.service → /lib/systemd/system/nginx.service.
root@server:~# systemctl status nginx
• nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor
   preset:
   enabled)
```

```
Active: active (running) since Wed 2021-01-06 21:16:21 UTC; 2 days ago
Docs: man:nginx(8)
Main PID: 659 (nginx)
Tasks: 2 (limit: 2282)
Memory: 11.6M
CGroup: /system.slice/nginx.service
        └─659 nginx: master process /usr/sbin/nginx -g daemon on;
master_process on;
           └─661 nginx: worker process

Jan 06 21:16:20 server systemd[1]: Starting A high performance web server
and a reverse proxy server...
Jan 06 21:16:21 server systemd[1]: Started A high performance web server
and a
```

Примечание! Сам файл юнита не изменяется.

Некоторые target являются только объединением в группу, тогда как другие могут использоваться для определения набора юнитов, которые необходимы системе для конечной загрузки, то есть перехода из нерабочего состояния в рабочее. У подобных targets есть одно специфическое свойство, которого нет у других: они могут быть изолированы. В системе всего таких четыре:

- **Emergency.target.** Запускается только минимальное количество юнитов, которого достаточно, чтобы исправить вашу систему в случае серьезных ошибок.
- **Rescue.target:** Запускаются все модули, необходимые для получения полностью работоспособной системы Linux. Однако запускаются востепенные службы, к примеру, sshd.service.
- **Multi-user.target:** Часто используется по умолчанию. Запускается все, что необходимо для полной функциональности системы, и обычно используется на серверах.
- **Graphical.target:** В дополнение к основным сервисам по возможности запускается и графический интерфейс.

Чтобы узнать, какая в данный момент используется опция, следует воспользоваться командой `systemctl get-default`, а установить с помощью `systemctl set-default`.

```
root@server:~# systemctl get-default
graphical.target
```

Управление опциями юнитов

С помощью команды `systemctl cat`, которая использовалась ранее, производится просмотр содержимого файлов, но они не всегда тому, что происходит в данный момент (`systemctl enable` влияло только на содержимое каталога `/etc/systemd/system`, но не на конфигурационный файл юнита). Чтобы выяснить, какие параметры доступны для конкретного юнита и применяются в данный момент, используйте команду `systemctl show`.

```
root@server:~# systemctl show sshd
Type=notify
Restart=on-failure
NotifyAccess=main
RestartUsec=100ms
TimeoutStartUsec=1min 30s
TimeoutStopUsec=1min 30s
TimeoutAbortUsec=1min 30s
RuntimeMaxUsec=infinity
WatchdogUsec=0
WatchdogTimestampMonotonic=0
RootDirectoryStartOnly=no
RemainAfterExit=no
GuessMainPID=yes
RestartPreventExitStatus=255
...
```

При изменении файлов модулей для применения опций необходимо убедиться, что изменения записаны в `/etc/systemd/system`, где должны быть созданы пользовательские файлы модулей. Рекомендуемый способ сделать это - использовать команду `systemctl edit`, которая создаст подкаталог для службы, которую вы редактируете. Например, при использовании `systemctl edit sshd.service` будет создан каталог с именем `/etc/systemd/systemd/ssh.service.d` и файлом `override.conf` внутри. Все настройки, применяемые в этом файле, заменяют все существующие настройки в служебном файле в `/usr/lib/systemd/system`.

```
root@server:~# cat /etc/systemd/system/ssh.service.d/override.conf
[Unit]
Description=Service for remote access over SSH protocol
root@server:~# systemctl status ssh.service
● ssh.service - Service for remote access over SSH protocol
   Loaded: loaded (/etc/systemd/system/ssh.service; enabled; vendor
  preset: enabled)
   Drop-In: /etc/systemd/system/ssh.service.d
   ...
```

Важно! При изменении опций необходимо не просто их описать, но и указать, к какой секции изменяемые параметры относятся. В примере обязательно указание `[Unit]` в совокупности с `Description`.

Создание юнита

Давайте попробуем создать свой простой юнит с названием `hello-world`. Как мы узнали ранее, для этого понадобится добавить файл **`hello-world.service`** в папку `/etc/systemd/system/`.

```
[Unit]
Description=Hello World Service
After=multi-user.target

[Service]
Type=simple
ExecStart=/usr/bin/dd if=/dev/zero

[Install]
WantedBy=multi-
```

Единственное назначение этой службы заключается в копировании (утилиты `dd`) “ничего” (`if=/dev/zero`) в “никуда” (`of=/dev/null`). По сути мы просто нагружаем операционную систему бесполезной работой. Запускаться служба должна в самом конце загрузки системы, поэтому устанавливаем параметрам `After` и `WantedBy` значение `multi-user.target`. На основе секции **[Install]** `systemd` будет создавать символические ссылки в необходимых папках.

Выполняем команду `systemctl daemon-reload` для перечитывания файлов и запускаем сервис.

```
root@server:~# systemctl daemon-reload
root@server:~# systemctl start hello-world.service
root@server:~# systemctl status hello-world.service
• hello-world.service - Hello World Service
   Loaded: loaded (/etc/systemd/system/hello-world.service; disabled;
 vendor preset: enabled)
   Active: active (running) since Sun 2021-01-10 14:29:00 UTC; 11min ago
 Main PID: 20462 (dd)
    Tasks: 1 (limit: 2282)
   Memory: 148.0K
    CGroup: /system.slice/hello-world.service
            └─20462 /usr/bin/dd if=/dev/zero of=/dev/null

Jan 10 14:29:00 server systemd[1]: Started Hello World Service.
root@server:~# systemctl enable hello-world
Created symlink /etc/systemd/system/multi-user.target.wants/hello-world.
service → /etc/systemd/system/hello-world.service.
```

Мониторинг процессов

Рассмотрим простейшие способы оценки состояния операционной системы. Для мониторинга работы процессов, запущенных в операционной системе (потребление памяти, ресурсов центрального процессора), нам потребуется ряд программ.

Одной из утилит, которую мы рассмотрим, и которая была использована ранее, является **ps**. Она покажет список запущенных процессов в операционной системе. Используя её в сочетании с командой **grep**, мы можем найти и получить полезную информацию о процессе.

```
root@server:~# ps -ef | grep -P
UID          PID     PPID  C  STIME TTY          TIME CMD
root           2         0  0   Jan08 ?        00:00:00 [kthreadd]
root          93         2  0   Jan08 ?        00:00:00 [ipv6_addrconf]
message+     601             1  0   Jan08 ?        00:00:01 /usr/bin/dbus-
--system     --address=systemd: --nofork --nopidfile --systemd-activation
--syslog-only
root        2046             1 99  14:28 ?        00:28:18 /usr/bin/dd
of=/dev/null
```

При поиске по имени команды мы получили несколько результатов, потому что шаблон поиска имеет совпадения и с другими процессами. Чтобы подобного не случалось, можно попробовать искать по PID, который в свою очередь можно найти с помощью утилиты **pgrep**.

```
root@server:~# pgrep -xl dd
20462 dd

root@server:~# ps -ef | grep -P '20462|CMD'
UID          PID     PPID  C  STIME TTY          TIME CMD
root        20462             1 99  14:28 ?        00:31:30 /usr/bin/dd
of=/dev/null
```

У **ps** существует большой набор ключей, которыми можно воспользоваться. К примеру, можно найти все запущенные пользовательские процессы, принадлежащие **user**.

```
root@server:~# ps -fU user
UID          PID     PPID  C  STIME TTY          TIME CMD
use         7953         1  0   14:17 ?        00:00:00 /lib/systemd/systemd --
r           7955      user 795  0   14:17 ?        00:00:00 (sd-pam)
use         8080         3  0   14:18 ?        00:00:00 sshd: user@pts/0
r           8081      795  0   14:18          00:00:00 -bash
```

Утилита **top** (table of process) выведет список и информацию о запущенных в системе процессах.

```
top - 15:50:27 up 2 days, 3:10, 1 user, load average: 0.28, 0.07, 0.19
Tasks: 107 total, 2 running, 105 sleeping, 0 stopped, 0 zombie
%Cpu(s): 54.5 us, 45.5 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1987.8 total, 433.0 free, 193.1 used, 1361.7 buff/cache
MiB Swap: 1706.0 total, 1705.5 free, 0.5 used. 1602.4 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
21004	root	20	0	5520	724	656	R	99.3	0.0	0:16.23	dd
1	root	20	0	105480	12984	8268	S	0.0	0.6	0:10.82	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.02	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-kblockd
9	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
10	root	20	0	0	0	0	S	0.0	0.0	0:01.09	ksoftirqd/0
11	root	20	0	0	0	0	I	0.0	0.0	0:30.08	rcu_sched
12	root	rt	0	0	0	0	S	0.0	0.0	0:00.98	migration/0
13	root	-51	0	0	0	0	S	0.0	0.0	0:00.00	idle_inject/0
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
16	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
17	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_tasks_kthre
18	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kauditd
19	root	20	0	0	0	0	S	0.0	0.0	0:00.05	khungtaskd
20	root	20	0	0	0	0	S	0.0	0.0	0:00.00	oom_reaper
21	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	writeback

Рис.1 - Вывод top

Утилита отобразит текущее состояние процессов в колонке S(tate).

- **(R)** Процесс в настоящее время активен и использует процессорное время или находится в очереди работоспособных процессов, ожидающих получения услуг.
- **(S)** Процесс ожидает завершения события.
- **(D)** Процесс находится в спящем состоянии, и его нельзя остановить. Обычно это происходит, когда процесс ожидает ввода-вывода.
- **(T)** Процесс был остановлен, что обычно происходит с интерактивным процессом оболочки, с помощью последовательности клавиш Ctrl-Z.
- **(Z)** Процесс был остановлен, но не может быть удален его родительским элементом, который перешел в неуправляемое состояние.

Важный параметр, который можно увидеть справа сверху, - это load average, который выражается как количество процессов, находящихся в рабочем состоянии (R) или в состоянии блокировки (D). Средняя нагрузка отображается за последние 1, 5 и 15 минут. **Важно!** Как показывает практика, средняя загрузка не должна превышать количество ядер CPU

в системе. Вы можете узнать количество ядер с помощью команды **lscpu**.

С помощью **top** легко увидеть процессы, на которые стоит обратить внимание. Исходя из рисунка 1, можно сделать вывод, что созданная ранее служба сильно нагружает систему, так как потребляет почти 100% CPU. С помощью нажатия на **k** можно передать сигнал прямо из **top**.

Выбираем номер процесса и номер сигнала для передачи.

```
PID to signal/kill [default pid = 20462]
  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
 20462 root        20   0   5520    780   720  R  93.8   0.0   55:00.66 dd

Send pid 20462 signal [15/sigterm]
  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
 20462 root        20   0   5520    780   720  R  93.8   0.0   55:00.66 dd
```

Состояние процесса следует оценивать не только по загрузке системы, но и по сообщениям, которые запущенные службы записывают. Процессы могут быть в нормальном с точки зрения потребления ресурсов состоянии, но в логах окажутся ошибки о невозможности выполнения заложенных действий. Ранее мы рассмотрели файлы логов, расположенные в каталоге **/var/log**, но помимо этого существует сервис **systemd-journald**, который хранит сообщения в журнале, двоичном файле **/run/log/journal**. Этот файл можно просмотреть с помощью команды **journalctl**.

Самый простой способ использовать **journalctl** - просто ввести команду.

```
root@server:~# journalctl
-- Logs begin at Mon 2020-12-07 17:59:17 UTC, end at Sun 2021-01-10
15:25:29 UTC. --
Dec 07 17:59:17 server kernel: Linux version 5.4.0-56-generic
(buildd@lgw01-amd64-025)(gcc version 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04))
#62-Ubuntu SMP Mon Nov 23 19:20:19 UTC 2020 (Ubuntu 5.4.0-56.62-g>
Dec 07 17:59:17 server kernel: Command
line: BOOT_IMAGE=/vmlinuz-5.4.0-56-generic root=/dev/mapper/ubuntu--
vg-ubuntu--lv ro Dec 07 17:59:17 server kernel: KERNEL supported cpus:
Dec 07 17:59:17 server Intel GenuineIntel
kernel: AMD AuthenticAMD
Dec 07 17:59:17 server Hygon HygonGenuine
kernel: Centaur CentaurHauls
Dec 07 17:59:17 server zhaoxi Shangha
kernel:
```

Поскольку журнал ведется с момента загрузки сервера, в начале отображаются сообщения, связанные с загрузкой. Если вы хотите увидеть по-

следние сообщения, которые были зарегистрированы, вы можете использовать `G` (в верхнем регистре), чтобы перейти в конец журнала, или опцию `-f` для просмотра наполнения журнала в реальном времени.

Одна из полезных функций - это возможность отсортировать ошибки.

```
root@server:~# journalctl -p err
-- Logs begin at Mon 2020-12-07 17:59:17 UTC, end at Sun 2021-01-10
15:25:29 UTC. --
Dec 07 17:59:17 server kernel: [drm:vmw_host_log [vmwgfx]] *ERROR*
Failed to send host log message.
Dec 07 17:59:17 server kernel: [drm:vmw_host_log [vmwgfx]] *ERROR*
Failed to send host log message.
Dec 08 07:52:51 server systemd-networkd-wait-online[3163]: Event loop
failed: Connection timed out
Dec 08 07:54:04 server systemd-networkd-wait-online[3609]: Event loop
failed: Connection timed out
...
```

Или сообщения, относящиеся к определенной службе.

```
root@server:~# journalctl -u nginx
-- Logs begin at Mon 2020-12-07 17:59:17 UTC, end at Sun 2021-01-10
15:25:29 UTC. --
Dec 17 17:24:17 server systemd[1]: Starting A high performance web server
and a reverse proxy server...
Dec 17 17:24:17 server systemd[1]: Started A high performance web server
and a reverse proxy server.
Dec 17 17:24:31 server systemd[1]: Stopping A high performance web server
and a reverse proxy server...
Dec 17 17:24:31 server systemd[1]: nginx.service: Succeeded.
Dec 17 17:24:31 server systemd[1]: Stopped A high performance web server
and a reverse proxy server.
Dec 17 17:26:41 server systemd[1]: Starting A high performance web server
and a reverse proxy server...
Dec 17 17:26:41 server systemd[1]: Started A high performance web server
and a reverse proxy server.
```

Важно! Журнал очищается при перезагрузке системы. Чтобы журнал оставался постоянным между перезагрузками, необходимо наличие каталога `/var/log/journal`, а так же параметр `Storage=auto` в `/etc/systemd/journal.conf`.

Практическое задание

1. Изменить конфигурационный файл службы SSH: `/etc/ssh/sshd_config`, отключив аутентификацию по паролю `PasswordAuthentication no`. Выполните рестарт службы `systemctl restart sshd (service sshd restart)`, верните аутентификацию по паролю, выполните `reload` службы `systemctl reload sshd (service sshd reload)`. В чём различие между действиями `restart` и `reload`?
2. Установить программу `mc` с помощью `apt-get install -y mc`. Запустите `mc`. Используя `ps`, найдите PID процесса, завершите процесс, передав ему сигнал `9`.
3. Создать юнит нового процесса с именем `userping`, который при старте будет выполнять `ping` адреса `127.0.0.1`. Сделать так, чтобы команда запускалась сразу при старте операционной системы. Запустить сервис с помощью `systemctl`, отследить `pid` процесса, остановить процесс с помощью сигналов в `top`.

Глоссарий

Загрузчик — программное обеспечение, обеспечивающее загрузку операционной системы сразу после включения компьютера или сервера.

Ядро — центральная часть операционной системы, обеспечивающая приложениям доступ к ресурсам компьютера (ресурсам центрального процессора, памяти и т. д.).

Initrd — виртуальная файловая система, используемая ядром ОС Linux перед монтированием файловых систем. На этой файловой системе могут находиться модули или какие-то особые параметры служб, которые необходимы для корректного старта ОС.

Systemd — система инициализации Linux, процесс для запуска юнитов (элементарных единиц в терминологии `systemd`) в Linux и управления ими в процессе работы системы. Его особенность — интенсивное распараллеливание запуска служб в процессе загрузки системы, что позволяет существенно ускорить запуск операционной системы.

Процесс — набор программного кода, выполняемого в памяти компьютера.

Приложение — программное обеспечение, включающее в себя исполняемый код, конфигурационные файлы, данные.

Таблица процессов — структура данных, описывающая все процессы, запущенные в данный момент в операционной системе, их PID, состояние, строку команды.

Дополнительные материалы

Процессы в Linux

Используемые источники

Загрузка операционной системы

Процессы в Linux

Робачевский А. Операционная система Unix

Костромин В. Linux для пользователя