

Процессы

Выполняющаяся программа называется в Linux процессом. Все процессы система регистрирует в таблице процессов, присваивая каждому уникальный номер — идентификатор процесса (process identifier, **PID**). Процессы получают доступ к ресурсам системы (оперативной памяти, файлам, внешним устройствам и т. п.) и могут изменять их содержимое. Доступ регулируется с помощью идентификатора пользователя и идентификатора группы, которые система присваивает каждому процессу. Процессы в Linux можно описать как контейнеры, в которых хранится вся информация о состоянии и выполнении программы

Например, программа инициализации, которая запускается сразу после завершения загрузки ядра тоже имеет свой процесс с идентификатором 0. Если программа работает хорошо, то все нормально, но если она зависла или вам нужно настроить ее работу может понадобиться управление процессами в Linux.

Каждый из процессов может находиться в одном из таких состояний:

Запуск — процесс либо уже работает, либо готов к работе и ждет, когда ему будет дано процессорное время;

Ожидание — процессы в этом состоянии ожидают какого-либо события или освобождения системного ресурса. Ядро делит такие процессы на два типа — те, которые ожидают освобождения аппаратных средств и приостановление с помощью сигнала;

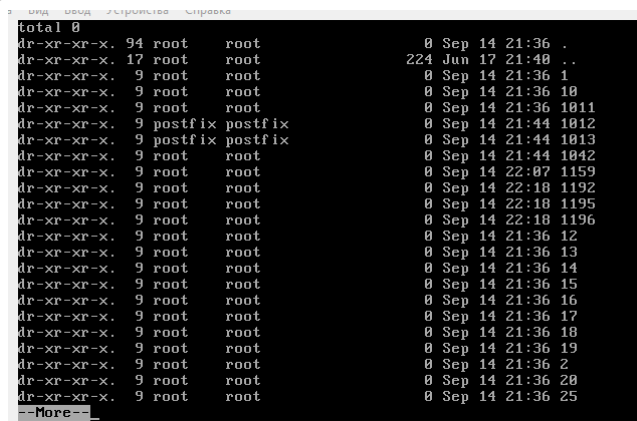
Остановлено — обычно, в этом состоянии находятся процессы, которые были остановлены с помощью сигнала;

Зомби — это мертвые процессы, они были остановлены и больше не выполняются, но для них есть запись в таблице процессов, возможно, из-за того, что у процесса остались дочерние процессы.

Каталог процессов

Информация о процессах находится в каталоге **/proc**

```
ls -al /proc | more
```



USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	1.3	128164	6824	?	Ss	21:36	0:02	/usr/lib/systemd/systemd --switched-root --system --deserialize 21
root	2	0.0	0.0	0	0	?	S	21:36	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S	21:36	0:00	[ksoftirqd/0]
root	5	0.0	0.0	0	0	?	S<	21:36	0:00	[kworker/0:0H]
root	6	0.0	0.0	0	0	?	S	21:36	0:00	[kworker/u2:0]
root	7	0.0	0.0	0	0	?	S	21:36	0:00	[migration/0]
root	8	0.0	0.0	0	0	?	S	21:36	0:00	[rcu_bh]
root	9	0.0	0.0	0	0	?	R	21:36	0:00	[rcu_sched]
root	10	0.0	0.0	0	0	?	S	21:36	0:00	[watchdog/0]
root	12	0.0	0.0	0	0	?	S	21:36	0:00	[kdevtmpfs]
root	13	0.0	0.0	0	0	?	S<	21:36	0:00	[netns]
root	14	0.0	0.0	0	0	?	S	21:36	0:00	[khungtaskd]
root	15	0.0	0.0	0	0	?	S<	21:36	0:00	[writeback]
root	16	0.0	0.0	0	0	?	S<	21:36	0:00	[kintegrityd]
root	17	0.0	0.0	0	0	?	S<	21:36	0:00	[bioset]
root	18	0.0	0.0	0	0	?	S<	21:36	0:00	[kblockd]
root	19	0.0	0.0	0	0	?	S<	21:36	0:00	[md]
root	20	0.0	0.0	0	0	?	S	21:36	0:01	[kworker/0:1]
root	25	0.0	0.0	0	0	?	S	21:36	0:00	[ksm]
root	26	0.0	0.0	0	0	?	SN	21:36	0:00	[kswapd0]
root	27	0.0	0.0	0	0	?	S<	21:36	0:00	[cryptol]
root	35	0.0	0.0	0	0	?	S<	21:36	0:00	[kthrotld]

Рис. ПР_03 Список всех процессов

Параметры ps

- A, (a) — выбрать все процессы;
- a — выбрать все процессы, кроме фоновых;
- d, (g) — выбрать все процессы, даже фоновые, кроме процессов сессий;
- N — выбрать все процессы кроме указанных;
- C — выбирать процессы по имени команды;
- G — выбрать процессы по ID группы;
- p, (p) — выбрать процессы PID;
- ppid — выбрать процессы по PID родительского процесса;
- s — выбрать процессы по ID сессии;
- t, (t) — выбрать процессы по tty;
- u, (U) — выбрать процессы пользователя.

Опции форматирования:

- c — отображать информацию планировщика;
- f — вывести максимум доступных данных, например, количество потоков;
- j, (j) — вывести процессы в стиле Jobs, минимум информации;
- M, (Z) — добавить информацию о безопасности;
- o, (o) — позволяет определить свой формат вывода;
- sort, (k) — выполнять сортировку по указанной колонке;
- L, (H) — отображать потоки процессов в колонках LWP и NLWP;
- m, (m) — вывести потоки после процесса;
- V, (V) — вывести информацию о версии;
- H — отображать дерево процессов linux;

Дочерний процесс

Системный вызов `fork()` («вилка, развилка»), порождает дочерний, процесс — точная копия породившего его родительского. Дочерний процесс ничем не отличается от родительского: имеет такое же окружение, те же стандартный ввод и стандартный вывод, одинаковое содержимое памяти.

Отличия:

- 1) процессы имеют разные PID, под которыми они зарегистрированы в таблице процессов;
- 2) различается возвращаемое значение `fork()` : родительский процесс получает в качестве результата `fork()` идентификатор процесса-потомка, а процесс-потомок получает "0".

Активный процесс (foreground process)

Процесс, имеющий возможность вводить данные с терминала. В каждый момент у каждого терминала может быть не более одного активного процесса.

Фоновый процесс (background process)

Процесс, запускаемый параллельно и не имеющий возможности вводить данные с терминала (только из файла). Пользователь может запустить любое, но не превосходящее заранее заданного в системе, число фоновых процессов

Пример ПР_03

Создать сценарий, имитирующий бесконечные вычисления и запустить его в фоновом режиме
Переход в домашний каталог

```
cd ~
    Создание файла сценария
vi loop
while true;
do true;
done
    сохранение
:wq
    Запуск в фоновом режиме (&)
sh loop&
```

Утилита top

Используется для просмотра запущенных процессов

Пример ПР_04

Вывести список процессов с помощью утилиты top

top

```
top - 22:57:34 up 1:21, 1 user, load average: 0.40, 0.90, 1.06
Tasks: 82 total, 2 running, 80 sleeping, 0 stopped, 0 zombie
%Cpu(s): 99.3 us, 0.3 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.3 si, 0.0 st
KiB Mem : 500152 total, 305020 free, 77352 used, 116972 buff/cache
KiB Swap: 839676 total, 839676 free, 0 used, 384076 avail Mem
```

PID	USER	PR	NI	UIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1349	root	20	0	113120	1192	1012	R	99.7	0.2	0:27.17	sh
1	root	20	0	128164	6824	4056	S	0.0	1.4	0:02.49	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kssoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworke/0:0H
6	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kworke/u2:0
7	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_sched
10	root	rt	0	0	0	0	S	0.0	0.0	0:00.04	watchdog/0
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
13	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	khungtaskd
15	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	writeback
16	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
17	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	bioset
18	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
19	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	md

Рис. ПР_04 Список процессов top

Информация о процессе:

- PID — идентификатор процесса
- USER — пользователь, от которого был запущен процесс
- PRI — приоритет процесса linux на уровне ядра (обычно NI+20)
- NI — приоритет выполнения процесса от -20 до 19
- S — состояние процесса
- CPU — используемые ресурсы процессора
- MEM — использованная память
- TIME — время работы процесса

На рис. ПР_04 в первой строке указан процесс, запущенный в примере ПР_03. В столбце %CPU указан процент использования центрального процессора (99,7) и статус R (Running - активный)

Пример ПР_05

Запустить второй экземпляр сценария loop в фоновом режиме и вывести список процессов

```
bash loop&
top
```

```
top - 23:04:32 up 1:20, 1 user, load average: 1.00, 1.34, 1.17
Tasks: 83 total, 3 running, 80 sleeping, 0 stopped, 0 zombie
%Cpu(s): 99.3 us, 0.7 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 500152 total, 305564 free, 77612 used, 116976 buff/cache
KiB Swap: 839676 total, 839676 free, 0 used, 383812 avail Mem
```

PID	USER	PR	NI	UIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1349	root	20	0	113120	1192	1012	R	49.8	0.2	6:22.94	sh
1373	root	20	0	113120	1180	1012	R	49.8	0.2	1:01.36	bash
1374	root	20	0	157504	2104	1504	R	0.7	0.4	0:00.29	top
1	root	20	0	128164	6824	4056	S	0.0	1.4	0:02.50	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kssoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworke/0:0H
6	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kworke/u2:0
7	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_sched
10	root	rt	0	0	0	0	S	0.0	0.0	0:00.04	watchdog/0
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
13	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	khungtaskd
15	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	writeback
16	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
17	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	bioset

Рис. ПР_05 Список процессов top

В настоящее время два процесса (sh и bash) делят процессор. Приоритеты обоих процессов равны 20 (столбец PR).

Другой вывод списка процессов (отсортированный по времени использования процессора. Знак "-" - сортировка по убыванию)

ps -aux --sort=-%cpu | more

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1349	79.1	0.2	113128	1192	tty1	R	22:57	7:14	sh loop
root	1373	49.9	0.2	113128	1188	tty1	R	23:02	1:52	bash loop
root	1	0.0	1.3	128164	6824	?	Ss	21:36	0:02	/usr/lib/systemd/systemd --switched-root --system --deserialize 21
root	2	0.0	0.0	0	0	?	S	21:36	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S	21:36	0:00	[ksoftirqd/0]
root	5	0.0	0.0	0	0	?	S<	21:36	0:00	[kworker/0:0H]
root	6	0.0	0.0	0	0	?	S	21:36	0:00	[kworker/u2:0]
root	7	0.0	0.0	0	0	?	S	21:36	0:00	[migration/0]
root	8	0.0	0.0	0	0	?	S	21:36	0:00	[rcu_bh]
root	9	0.0	0.0	0	0	?	R	21:36	0:00	[rcu_sched]
root	10	0.0	0.0	0	0	?	S	21:36	0:00	[watchdog/0]
root	12	0.0	0.0	0	0	?	S	21:36	0:00	[kdevtmpfs]
root	13	0.0	0.0	0	0	?	S<	21:36	0:00	[netns]
root	14	0.0	0.0	0	0	?	S	21:36	0:00	[khungtaskd]
root	15	0.0	0.0	0	0	?	S<	21:36	0:00	[writeback]
root	16	0.0	0.0	0	0	?	S<	21:36	0:00	[kintegrityd]
root	17	0.0	0.0	0	0	?	S<	21:36	0:00	[bioset]
root	18	0.0	0.0	0	0	?	S<	21:36	0:00	[kblockd]
root	19	0.0	0.0	0	0	?	S<	21:36	0:00	[md]
root	25	0.0	0.0	0	0	?	S	21:36	0:00	[kswapd0]
root	26	0.0	0.0	0	0	?	SN	21:36	0:00	[ksmd]
root	27	0.0	0.0	0	0	?	S<	21:36	0:00	[cryptol]
--More--										

Рис. ПР_05_b Список процессов ps

Пример ПР_06

Вывести оба фоновых процесса из фонового режима и остановить их выполнение

Решение:

1 . Выводим последний процесс з фонового

fg

```
[root@localhost ~]# fg
bash loop
```

2. Останавливаем выполнение

CTRL+C

3. Аналогично для второго процесса

4. Проверка

top

top - 20:38:56 up 20 min, 1 user, load average: 0.00, 0.04, 0.12										
Tasks: 81 total, 1 running, 80 sleeping, 0 stopped, 0 zombie										
Cpu(s): 0.0 us, 0.3 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st										
Mem: 500152 total, 305536 free, 78704 used, 115912 buff/cache										
Mem Swap: 839676 total, 839676 free, 0 used, 382672 avail Mem										
PID	USER	PR	NI	UIRT	RES	SHR	S	%CPU	%MEM	TIME+ COMMAND
903	root	20	0	562392	16596	5908	S	0.3	3.3	0:00.39 tuned
1085	root	20	0	157584	2100	1504	R	0.3	0.4	0:00.11 top
1	root	20	0	128164	6824	4056	S	0.0	1.4	0:00.06 systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00 kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.03 ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 kworker/0:0H
6	root	20	0	0	0	0	S	0.0	0.0	0:00.00 kworker/u2:0
7	root	rt	0	0	0	0	S	0.0	0.0	0:00.00 migration/0
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00 rcu_bh
9	root	20	0	0	0	0	S	0.0	0.0	0:00.17 rcu_sched
10	root	rt	0	0	0	0	S	0.0	0.0	0:00.01 watchdog/0
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00 kdevtmpfs
13	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 netns
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00 khungtaskd
15	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 writeback
16	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 kintegrityd
17	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 bioset
18	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 kblockd

Запущенные сценарии loop в списке процессов отсутствуют.

Приоритет процесса

Приоритет процесса linux означает, насколько больше процессорного времени будет отдано этому процессу по сравнению с другими. Можно очень тонко настроить какая программа будет работать быстрее, а какая медленнее. Значение приоритета может колебаться от 19 (минимальный приоритет) до -20 — максимальный приоритет процесса linux. Причем, уменьшать приоритет можно с правами обычного пользователя, но чтобы его увеличить нужны права суперпользователя.

Команда nice

Указывает приоритет запускаемого процесса. Команда **nice** запускает программу с изменённым приоритетом для планировщика задач. Слово «**nice**» в английском языке обозначает, в частности, «вежливый». По этимологии этой команды процесс с большим значением **nice** — более

вежлив к другим процессам, позволяя им использовать больше процессорного времени, поскольку он сам имеет меньший приоритет (и, следовательно, большее «значение вежливости» — niceness value).

Синтаксис

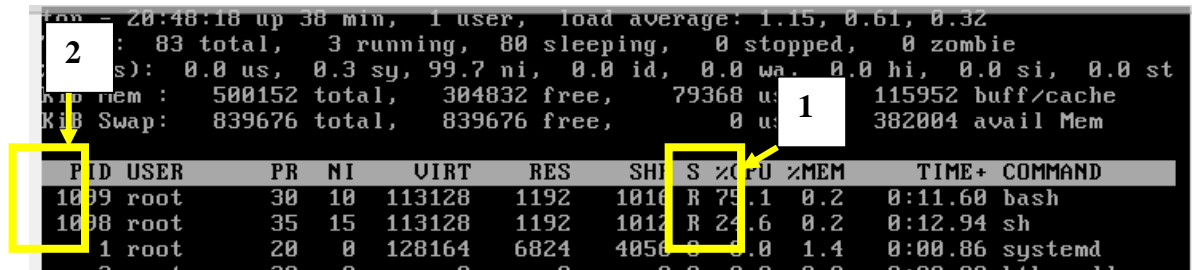
nice -n значение_nice команда

Пример ПР_07

Запустить 2 сценария loop в фоновом режиме, первый - с приоритетом 15, второй с приоритетом 10.

Решение:

```
nice -n 15 sh loop&
nice -n 10 bash loop&
top
```



PID	USER	PR	NI	UIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1099	root	30	10	113128	1192	1016	R	75.1	0.2	0:11.60	bash
1098	root	35	15	113128	1192	1012	R	24.6	0.2	0:12.94	sh
1	root	20	0	128164	6824	4056	S	0.0	1.4	0:00.86	systemd

Рис. ПР_07 Проверка приоритета запущенного процесса

Из вывода команды **top** (рис. ПР_07, 1) видно, что процессу **bash loop&** выделяется процессорного времени больше чем **sh loop&**.

Изменить приоритет уже запущенного процесса (renice)

renice -n значение_nice id_процесса

Пример ПР_08

Для запущенных в примере ПР_07 сценариев изменить значения приоритетов:

sh loop& - приоритет 8

bash loop& - приоритет 12.

Решение:

1. Из вывода команды **top** определяем PID процессов (рис. ПР_07, 2)

!!! **ВНИМАНИЕ** Вывод у каждого пользователя будет свой, поэтому номера будут различны

2. Меняем приоритет первого процесса (sh, его PID=1098)

```
renice -n 8 -p 1098
```

```
[root@localhost ~]# renice -n 8 -p 1098
1098 (process ID) old priority 15, new priority 8
[root@localhost ~]#
```

```
renice -n 12 -p 1099
```

```
top
```

Пример ПР_09

Для всех процессов пользователя **user** установить приоритет 19

Решение:

```
renice -n 19 -u user
```

Выделяя приоритет нужно отдавать себе отчёт в том, что легко можно "убить" систему, случайно выделив все ресурсы на выполнение «тяжелых» задач так, что вмешаться уже не будет возможности. А добавить к этому ещё и возможные опечатки в скрипте, которые могут стать фатальными. Будьте внимательны!

Самостоятельно

Ознакомиться с командой **ionice**

Сигналы

Сигналы - это средство, с помощью которого процессам можно передать сообщения о некоторых событиях в системе. Сами процессы тоже могут генерировать сигналы, с помощью которых они передают определенные сообщения ядру и другим процессам. С помощью сигналов можно осуществлять такие акции управления процессами, как *приостановка процесса, запуск приостановленного процесса, завершение работы процесса*. Всего в Linux существует 63 разных сигнала, их перечень можно посмотреть по команде.

```
kill -l
```

Сигналы принято обозначать номерами или символическими именами. Все имена начинаются на **SIG**, но эту приставку иногда опускают: например, сигнал с номером 1 обозначают или как **SIGHUP**, или просто как **HUP**.

Когда процесс получает сигнал, то возможен один из двух вариантов развития событий. Если для данного сигнала определена подпрограмма обработки, то вызывается эта подпрограмма. В противном случае ядро выполняет от имени процесса действие, определенное по умолчанию для данного сигнала. Вызов подпрограммы обработки называется перехватом сигнала. Когда завершается выполнение подпрограммы обработки, процесс возобновляется с той точки, где был получен сигнал.

Можно заставить процесс игнорировать или блокировать некоторые сигналы. Игнорируемый сигнал просто отбрасывается процессом и не оказывает на него никакого влияния. Блокированный сигнал ставится в очередь на выдачу, но ядро не требует от процесса никаких действий до разблокирования сигнала. После разблокирования сигнала программа его обработки вызывается только один раз, даже если в течение периода блокировки данный сигнал поступал несколько раз.

Таблица ПР_01. Сигналы

N	ИМЯ	ОПИСАНИЕ	МОЖНО ПЕРЕХВАТЫВАТЬ	МОЖНО БЛОКИРОВАТЬ	КОМБИНАЦ ИЯ КЛАВИШ
1	HUP	Hangup. Отбой	Да	Да	
2	INT	Interrupt. В случае выполнения простых команд вызывает прекращение выполнения, в интерактивных программах - прекращение активного процесса	Да	Да	⌘+C или ⌘
3	QUIT	Как правило, сильнее сигнала Interrupt	Да	Да	⌘+⌘
4	ILL	Illegal Instruction. Центральный процессор столкнулся с незнакомой командой (в большинстве случаев это означает, что допущена программная ошибка). Сигнал отправляется программе, в которой возникла проблема	Да	Да	
8	FPE	Floating Point Exception. Вычислительная ошибка, например, деление на ноль	Да	Да	
9	KILL	Всегда прекращает выполнение процесса	Нет	Нет	
11	SEGV	Segmentation Violation. Доступ к недозволённой области памяти	Да	Да	
13	PIPE	Была предпринята попытка передачи данных с помощью конвейера или	Да	Да	

		очереди FIFO, однако не существует процесса, способного принять эти данные			
15	TERM	Software Termination. Требование закончить процесс (программное завершение)	Да	Да	
17	CHLD	Изменение статуса порожденного процесса	Да	Да	
18	CONT	Продолжение выполнения приостановленного процесса	Да	Да	
19	STOP	Приостановка выполнения процесса	Нет	Нет	
20	TSTR	Сигнал останова, генерируемый клавиатурой. Переводит процесс в фоновый режим	Да	Да	<Ctrl>+<Z>

Некоторые сигналы можно сгенерировать с помощью определенных комбинаций клавиш. Но такие комбинации существуют не для всех сигналов. Зато имеется команда **kill**, которая позволяет послать заданному процессу любой сигнал. Как уже было сказано, с помощью этой команды можно получить список всех возможных сигналов, если указать опцию -l. Если после этой опции указать номер сигнала, то будет выдано его символическое имя, а если указать имя, то получим соответствующий номер.

Для послыки сигнала процессу (или группе процессов) можно воспользоваться командой kill в следующем формате:

```
kill [-сигн] PID [PID..]
```

где сигн - это номер сигнала, причем если указание сигнала опущено, то посылается сигнал **15** (TERM - программное завершение процесса). **Чаще всего используется сигнал 9 (KILL), с помощью которого суперпользователь может завершить любой процесс. Но сигнал этот очень "грубый", если можно так выразиться, поэтому его использование может привести к нарушению порядка в системе. Поэтому в большинстве случаев рекомендуется использовать сигналы TERM или QUIT, которые завершают процесс более "мягко".**

Естественно, что наиболее часто команду kill вынужден применять суперпользователь. Он должен использовать ее для уничтожения процессов-зомби, зависших процессов (они показываются в листинге команды ps как <exiting>), процессов, которые занимают слишком много процессорного времени или слишком большой объем памяти и т. д. Особый случай - процессы, запущенные злоумышленником.

Пример ПР 10

Определить PID сценария **sh loop&** и приостановить его выполнение

Решение:

1. Вывод списка процессов с применением фильтра **grep**

ps -aux | grep loop

```
[root@localhost ~]# ps -aux | grep loop
root    1155  8.0  0.2 113128  1188 tty1      R   21:33   0:36 sh loop
root    1156  91.6  0.2 113128  1192 tty1      R   21:34   6:45 bash loop
root    1206   0.0  0.1 112660   972 tty2      R+  21:41   0:00 grep --color=au
to loop
[root@localhost ~]#
```

Рис. ПР_10_1

Искомый **PID=1155**

2. Приостановка процесса (сигнал STOP имеет номер 19)

```
kill 19 1098
```

3. Проверка

```
ps 1155
```

```
[root@localhost ~]# ps 1155
  PID TTY          STAT TIME COMMAND
 1155 tty1          T      1:10 sh loop
[root@localhost ~]#
```

Рис. ПР_10_2 Состояние процесса

Статус процесса **T** (рис. ПР_10_2)

Пример ПР 11

Вывести список приостановленных процессов и запустить на выполнение приостановленный в примере ПР_10 сценарий **sh loop&**

Решение:

1. Вывод списка процессов

ps -aux --sort=stat | grep T

```
[root@localhost ~]# ps -aux --sort=stat | grep T
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      1155  6.4  0.2 113128 1188 tty1          T    21:33   1:10 sh loop
root      1257  0.0  0.1 112660  952 tty2      S+   21:52   0:00 grep --color=au
to T
```

Рис. ПР_11_1 ps

или

top -o S

```
top - 21:55:56 up 1:45, 2 users, load average: 1.01, 1.07, 1.24
Tasks: 88 total, 2 running, 85 sleeping, 1 stopped, 0 zombie
%Cpu(s): 99.7 us, 0.3 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 500152 total, 296164 free, 83404 used, 120584 buff/cache
KiB Swap: 839676 total, 839676 free, 0 used, 377488 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
 1155 root      20   0 113128  1188  1012 T   0.0   0.2   1:10.55 sh
 1 root      20   0 128164  6824  4056 S   0.0   1.4   0:00.90 systemd
 2 root      20   0 0        0      0 S   0.0   0.0   0:00.00 kthreadd
 3 root      20   0 0        0      0 S   0.0   0.0   0:00.04 ksoftirqd/0
```

Рис. ПР_11_2 top

2. Запуск процесса

kill -18 1155

3. Проверка

top

```
  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
 1156 root      20   0 113128  1192  1012 R  49.8   0.2   24:10.89 bash
 1155 root      20   0 113128  1188  1012 R  49.5   0.2   1:56.96 sh
 1261 root      20   0 157584  2108  1504 R   0.3   0.4   0:00.21 top
 1 root      20   0 128164  6824  4056 S   0.0   1.4   0:00.90 systemd
 2 root      20   0 0        0      0 S   0.0   0.0   0:00.00 kthreadd
```

Рис. ПР_11_3 Список процессов

Процесс снова запущен

Пример ПР 12

Завершить сценарий **bash loop&** (PID=1156, рис. ПР_11_3)

Решение:

kill 1156

Проверку осуществите самостоятельно

Пример ПР 13

Подсчитать количество запущенных процессов

Решение:

1. Будет использоваться конвейер (**|**) из команд **ps** и **wc** (рис. ПР_13)

ps -aux | wc -l

```
[root@localhost ~]# ps -aux | wc -l
90
```

Рис. ПР_13

Команда **ps** с ключом **-aux** вывела по одной строке для каждого процесса, а утилита **wc** с ключом **-l** подсчитала количество этих строк. Количество процессов равно 90.