ПРОЦЕССЫ В ОС LINUX

ПРОЦЕСС И ЕГО СОСТАВЛЯЮЩИЕ

Процесс — это абстракция, используемая для описания выполняющийся программы. Процесс представляет из себя системный объект, посредством которого можно контролировать обращение программы к памяти, центральному процессору и ресурсам ввода-вывода. В операционных системах Linux и Unix системные и пользовательские процессы подчиняются одним и тем же правилам, благодаря чему управление осуществляется с помощью единого набора команд.

Процесс состоит из адресного пространства и набора структур данных, содержащихся внутри ядра. Адресное пространство представляет собой совокупность страниц памяти, которые были выделены ядром для выполнения процесса. В него загружается код и используемые им библиотеки функций, а также переменные, содержимое стеков и различная вспомогательная информация, необходимая ядру для работы процесса. Поскольку в системах UNIX и Linux поддерживается концепция виртуальной памяти, страницы адресного пространства процесса в конкретный момент времени могут находится либо в физической памяти, либо в разделе подкачки, т.е. на диске.

В структуре данных ядра хранится всевозможная информация о каждом процессе. К наиболее важным относят:

- Таблицу распределения памяти
- Текущий статус (неактивен, приостановлен, выполняется и т.п.)
- Приоритет
- Информацию об используемых ресурсах
- Информацию о файлах и сетевых портах, открытых процессом
- Маску сигналов (запись о том, какие сигналы блокируются)
- Имя владельца процесса

Поток выполнения, обычно именуемой просто потоком, представляет результат разветвления в выполнении процесса. Поток наследует многие атрибуты своего процесса, причем в рамках одного процесса могут выполняться одновременно (параллельно) несколько потоков — такая модель выполнения получила название многопоточности. В старых однопроцессорных системах параллельное выполнение моделируется ядром, но в мультиядерных и многопроцессорных архитектурах потоки могут выполняться одновременно в различных ядрах. Такие многопоточные приложения, как BIND и Apache, извлекают максимальную пользу из мультиядерных систем, поскольку эти приложения могут обрабатывать несколько запросов одновременно.

АТРИБУТЫ ПРОЦЕССА

Ядро назначает каждому процессу уникальный идентификатор **PID**. PID — Proccess ID. Большинство команд и системных вызовов, работающих с процессами, требуют указания конкретного идентификатора, чтобы был ясен контекст операции. Идентификаторы присваиваются по порядку по мере создания процессов.

Ни в UNIX, ни в Linux нет системного вызова, который бы инициировал новый процесс для выполнения конкретной программы. Для того, чтобы породить новый процесс, существующий процесс должен клонировать себя сам. Клон может заменить выполняемую программу другой.

В операции клонирования исходный процесс называют родительским, а его клон — дочерним. Помимо собственного идентификатора, каждый дочерний процесс имеет атрибут PPID (Parent Proccess ID), который совпадает с идентификатором породившего его дочернего процесса. Стоит отметить, что PPID — весьма полезная информация, если приходится иметь дело с неизвестными процессами. Отслеживание истоков процесса может облегчить понимание его назначения и значимости.

Когда система загружается, ядро самостоятельно запускает несколько процессов. Наиболее важный из них — демон *init*, идентификатор которого всегда равен 1. Демон *init* отвечает за выполнение сценариев запуска системы. Все процесса, кроме тех, что создаются ядром, **являются потомками демона init.**

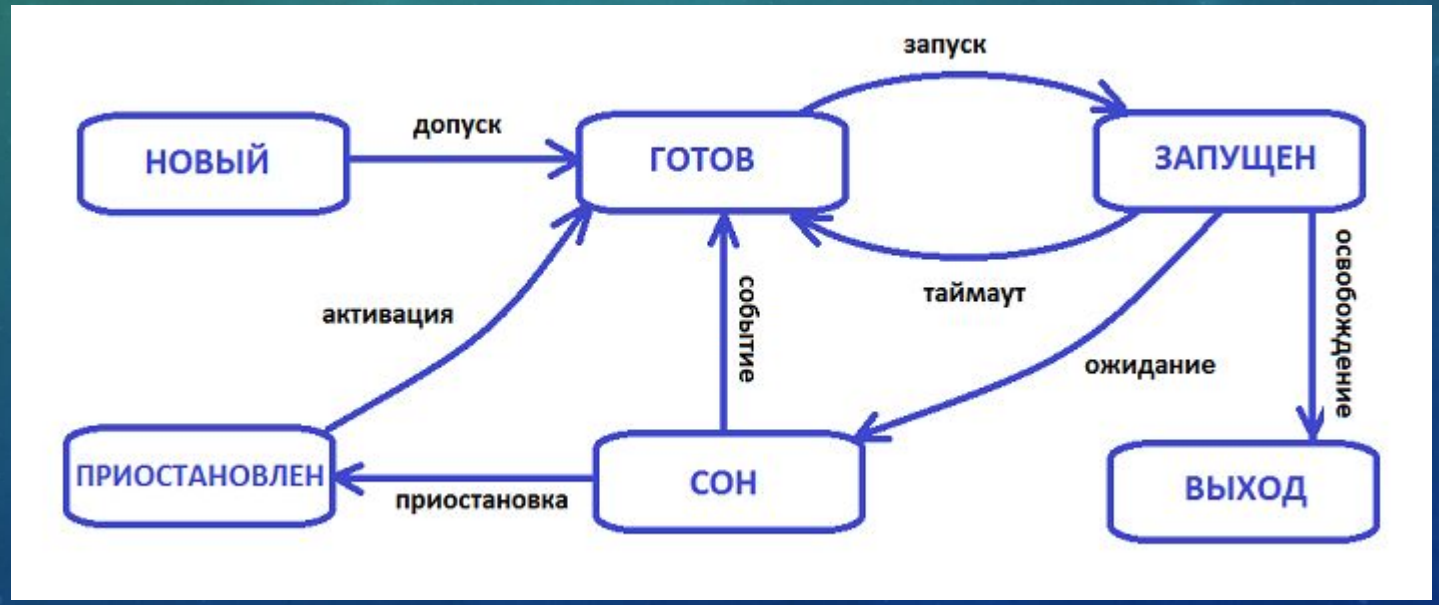
UID (User ID) — это идентификатор пользователя, создавшего данный процесс. Менять атрибуты процесса могут только его создатель (владелец) и суперпользователь. EUID (Effective User ID) — это текущий пользовательский идентификатор процесса, предназначенный для того, чтобы определить, к каким ресурсам и файлам у процесса есть доступ в данный момент. У большинства программ значения UID и EUID одинаковы. Исключение составляют программы, у которых установлен бит смены идентификатора пользователя (setuid).

GID (Group ID) — это идентификатор группы, к которому принадлежит владелец процесса. Текущий идентификатор группы (EGID) связан с атрибутом GID так же, как и значение EUID связано с UID.

Приоритет процесса определяет, какую долю времени центрального процессора получает программа. Ядро применяет динамический алгоритм вычисления приоритетов, учитывающий сколько времени центрального процессора уже использовал процесс и сколько времени он ожидает в своей очереди.

ЖИЗНЕННЫЙ ЦИКЛ ПРОЦЕССА

Создание процесса — это переход процесса из состояния «Новый» в состояние «Готов». В момент создания процесса операционная система подготавливает структуру данных для него. Новому процессу присваивается собственный PID, и учет ресурсов ведется независимо от предка. Тот факт, что процесс существует — еще не дает ему права на использование ресурсов центрального процессора.



Готовый процесс получил все необходимые ресурсы и ждет, пока системный планировщик предоставит ему доступ к центральному процессору. При выделении доступа процесс запускается и переходит в активное состояние (выполняется).

Из состояния «Запущен» есть два пути:

- Таймаут — процесс поработал и освободил ресурсы процессора (перешел в очередь)
- Ожидание — процесс переведен в режим ожидания, где ждет некоего сигнала

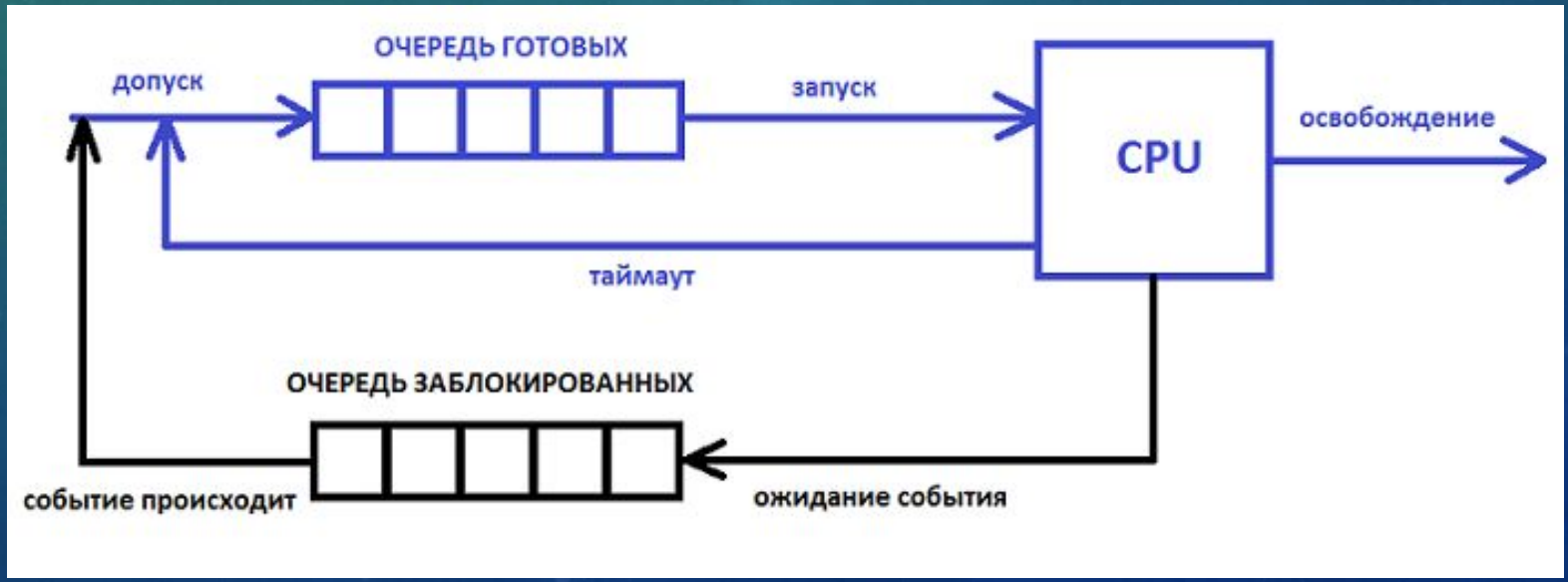
Если процесс осуществил системной вызов, который нельзя завершить немедленно, то ядро переводит его в режим ожидания (сон). Ожидающий процесс ждет наступления определенного события, будь то поступление данных с терминала или из сетевого соединения. Многие системные демоны проводят в этом состоянии большую часть своего времени. Важно отметить, что в данном случае процесс будет продолжать храниться в оперативной памяти.

Некоторые операции переводят процесс в состояние непрерывного ожидания (приостановлен). В данном состоянии процесс ожидает определенного сигнала от аппаратной части и не реагирует на другие сигналы. При этом процесс выгружен из оперативной памяти на жесткий диск (swap-раздел). Для того, чтобы избавиться от такого процесса необходимо устранить породившую их проблему или перезагрузить систему.

После того как процесс завершился — он уничтожается.

Зомби — это процесс, который закончил выполняться, но информация об этом еще не поступила родительскому процессу. Процесс при завершении высвобождает все свои ресурсы (за исключением PID) и становится «зомби» — пустой записью в таблице процессов, хранящий код завершения для родительс оцесса.

Все процессы выстраиваются в очередь на выполнение команд. При этом существует несколько очередей в зависимости от статуса процесса: очередь готовых и очередь заблокированных. По истечении работы процесс переходит в начало очередь и вновь ждет своего момента для доступа к ресурсам центрального процессора.



РАБОТА С ПРОЦЕССАМИ В LINUX

Просмотреть список всех процессов, выполняемых в текущий момент времени можно с помощью команды *ps*. С помощью команды *ps* можно получить информацию об идентификаторах, приоритете и управляющем терминале того или иного процесса. Она также позволяет выяснить объем оперативной памяти, который использует процесс, сколько времени центрального процессора заняло его выполнение, а также состояние процесса (выполняется, остановлен, простаивает и т.д.).

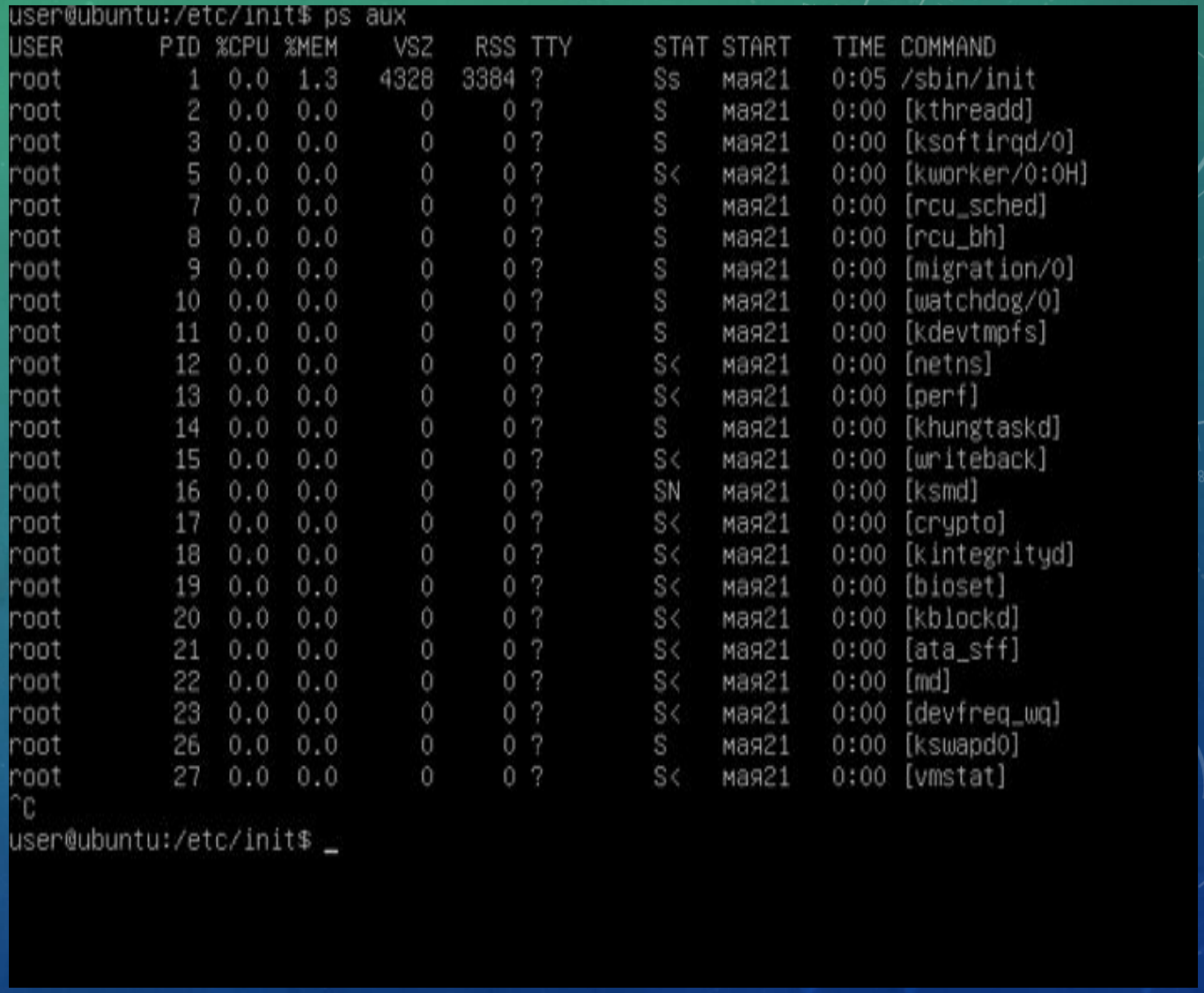
Получить список всех процессов можно с помощью следующей команды:

- *user@ubuntu$ ps aux*

Ключ *a* используется для вывода всех процессов, ключ *x* — отображает процессы, отсоединенные от терминала, ключ *u* — обеспечивает фильтрование по имени или идентификатору пользователя, который запустил программу.

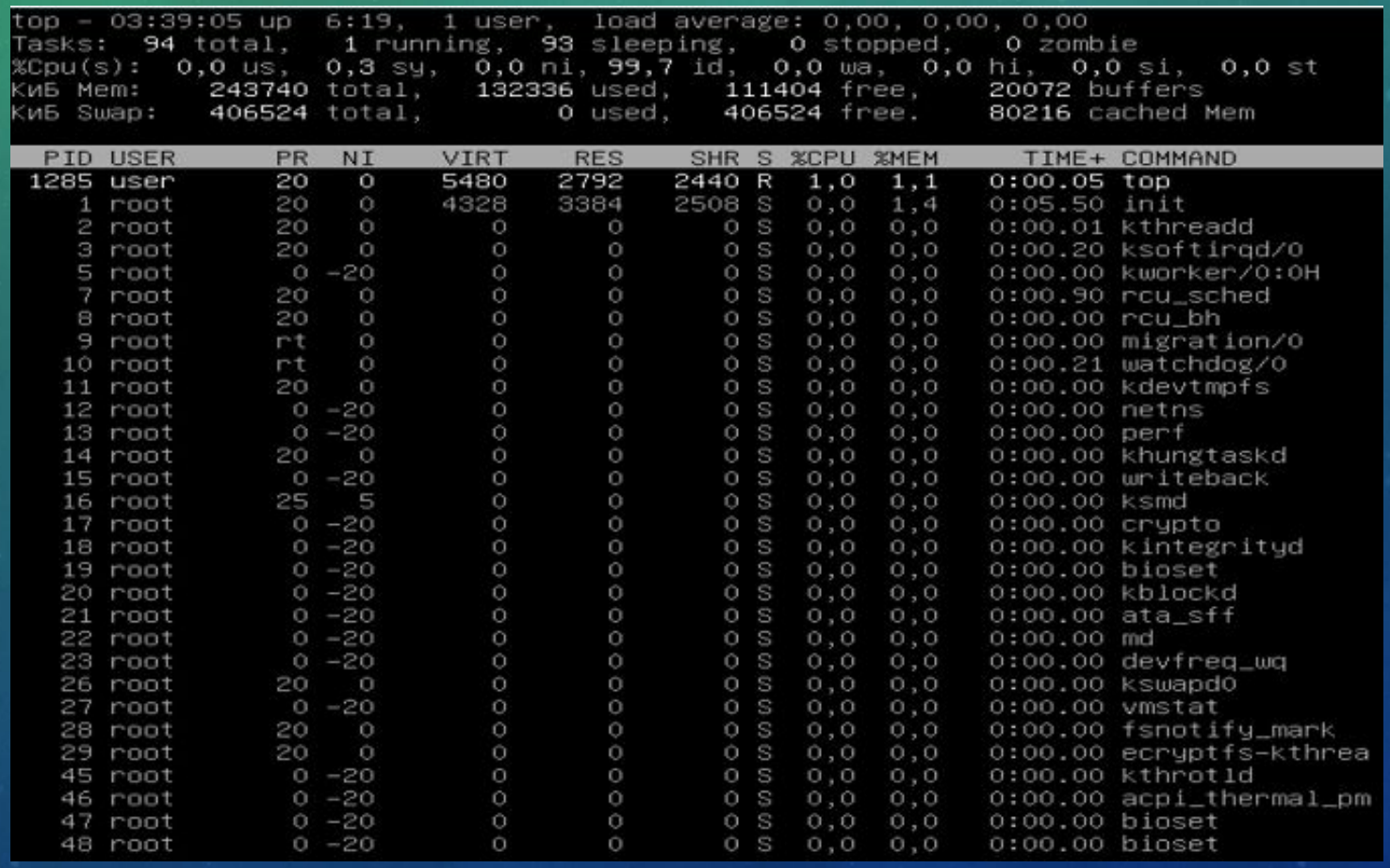
Значение столбцов при выводе команды *ps aux*:

- USER — имя владельца процесса
- PID — идентификатор процесса
- %CPU — доля времени центрального процессора, которая тратится на данный процесс (в процентах)
- %MEM — часть реальной памяти, которая тратится на данный процесс (в процентах)
- VSZ — виртуальный размер процесса
- RSS — количество страниц памяти
- TTY — идентификатор управляющего терминала
- STAT — текущий статус процесса (R-выполняется, D-ожидает записи на диск, S-неактивен, T-приостановлен, Z-зомби)
- TIME — количество времени центрального процессора, затраченное на выполнение данного процесса
- COMMAND — имя и аргументы команды



Команда ps позволяет сделать только разовый «снимок» системы. Для динамического отслеживания процессов используют команду *top*.

Наиболее активные процессы находятся вверху. Команда top отображает также статистику по статусам процессов, объем используемых ресурсов ЦПУ и оперативной памяти.



```
top - 03:39:05 up  6:19,  1 user,  load average: 0,00, 0,00, 0,00
Tasks:  94 total,   1 running,  93 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
КиБ Mem:    243740 total,   132336 used,   111404 free,    20072 buffers
КиБ Swap:   406524 total,        0 used,   406524 free.    80216 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1285 user      20   0    5480   2792   2440 R   1,0  1,1   0:00.05 top
    1 root      20   0    4328   3384   2508 S   0,0  1,4   0:05.50 init
    2 root      20   0       0      0      0 S   0,0  0,0   0:00.01 kthreadd
    3 root      20   0       0      0      0 S   0,0  0,0   0:00.20 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S   0,0  0,0   0:00.90 rcu_sched
    8 root      20   0       0      0      0 S   0,0  0,0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S   0,0  0,0   0:00.00 migration/0
   10 root      rt   0       0      0      0 S   0,0  0,0   0:00.21 watchdog/0
   11 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kdevtmpfs
   12 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 netns
   13 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 perf
   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      25   5       0      0      0 S   0,0  0,0   0:00.00 ksmd
   17 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 crypto
   18 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 kintegrityd
   19 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 bioset
   20 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 kblockd
   21 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 ata_sff
   22 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 md
   23 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 devfreq_wq
   26 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kswapd0
   27 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 vmstat
   28 root      20   0       0      0      0 S   0,0  0,0   0:00.00 fsnotify_mark
   29 root      20   0       0      0      0 S   0,0  0,0   0:00.00 ecryptfs-kthrea
   45 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 kthrotld
   46 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 acpi_thermal_pm
   47 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 bioset
   48 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 bioset
```

«Сигналы — это запросы на прерывания, реализуемые на уровне процессов»

Когда поступает сигнал, возможен один из двух вариантов событий. Если процесс назначил сигналу подпрограмму обработки, то после вызова ей предоставляется информация о контексте, в котором был сгенерирован сигнал. В противном случае ядро выполняет от имени процесса действия, заданные по умолчанию. Эти действия зависят от сигнала, а в некоторых случаях еще создается дамп памяти.

«Дамп памяти — это файл, содержащий образ памяти процесса»

Процедура вызова обработчика называется перехватом сигнала. Когда выполнение обработчика завершается, процесс возобновляется с той точки, где был получен сигнал.

Для того, чтобы некоторые сигналы не поступали в программу можно задать их игнорирование и блокирование. Игнорируемый сигнал просто пропускается и не влияет на работу процессора. Блокируемый сигнал ставится в очередь на обработку, но ядро не требует от процесса никаких действий до явного разблокирования сигнала.

Определено свыше тридцати разных сигналов, и они находят самое разное применение. Самые распространенные из них:

- KILL — безусловное завершение процесса на уровне ядра
- STOP — приостанавливает выполнение процесса
- CONT — возобновляет выполнение процесса
- TSTP — генерируется при нажатии CTRL + Z, приостанавливает процесс пользователем
- TERM — запрос на завершение программы (процесс осуществляет очистку и нормально завершается)
- QUIT — то же самое, что и TERM + создается дамп памяти
- HUP — команда сброса
- BUS — ошибка на шине (указывает на неправильное обращение к памяти)
- SEGV — ошибка на сегментации (указывает на неправильное обращение к памяти)

Сигналы KILL и STOP нельзя ни перехватить, ни заблокировать, ни проигнорировать.

Команда *kill* используется для отправки сигналов процессу. Kill имеет следующий синтаксис:

- *user@ubuntu$ kill [-сигнал] PID*