**10. Процессы**

Современные операционные системы обычно являются *многозадачными*, в том смысле, что создают иллюзию одновременного решения множества задач, быстро переключаясь с выполнения одной программы на другую. Ядро Linux управляет всем этим посредством *процессов*. Именно с помощью процессов Linux организует приостановку программ в ожидании, пока наступит их очередь использовать процессор.

Иногда компьютер становится «вялым», или приложение вообще перестает откликаться на команды пользователя. Сейчас мы познакомимся с некоторыми инструментами командной строки, позволяющими увидеть, что делают программы, и завершить процессы, вышедшие из-под контроля.

В этой главе будут представлены следующие команды:

• ps — выводит список процессов, выполняющихся в текущий момент.

• top — выводит задачи.

• jobs — выводит список активных заданий.

• bg — переводит задание в фоновый режим работы.

• fg — переводит задание в режим работы на переднем плане.

• kill — посылает сигнал процессу.

• killall — останавливает процессы по именам.

• shutdown — останавливает или перезагружает систему.

**Как действует процесс**

В момент запуска системы ядро инициирует выполнение нескольких собственных задач в виде процессов и запускает программу с названием init. В свою очередь init выполняет последовательность сценариев командной оболочки (находятся в */etc*), называемых *сценариями начальной загрузки* (init scripts), которые запускают все системные службы. Многие из этих служб реализованы как *программы-демоны* (daemon programs), то есть программы, действующие в фоновом режиме и выполняющие свою работу без участия пользователя. Поэтому, даже в отсутствие зарегистрированных пользователей система выполняет определенные служебные процедуры.

Принцип, по которому программа может запускать другие программы, выражается правилом: *родительский процесс* запускает *дочерний процесс*.

Ядро хранит информацию обо всех процессах, чтобы упорядочить их работу. Например, каждому процессу присваивается номер, который называют *идентификатором процесса* (Process ID, PID). Идентификаторы процессов присваиваются в порядке возрастания, при этом процесс init всегда получает идентификатор PID 1. Ядро также следит за памятью, выделенной каждому процессу, и за готовностью процессов возобновить выполнение. Подобно файлам, процессы также имеют идентификаторы владельца и пользователя, эффективный (или действующий) идентификатор пользователя и т.д.

**Просмотр списка процессов с помощью ps**

Чаще всего для просмотра списка процессов используется команда ps. Программа ps имеет множество параметров, но в самом простейшем случае она используется следующим образом:

[me@linuxbox ~]$ **ps**

  PID TTY          TIME CMD

5198 pts/1    00:00:00 bash

10129 pts/1    00:00:00 ps

В этом примере команда вывела список с двумя процессами: процесс 5198 и процесс 10129 — программы bash и ps соответственно. Как можно заметить, по умолчанию ps выводит не очень много информации, только процессы, связанные с текущим сеансом. Чтобы увидеть больше, следует передать дополнительные параметры, но прежде чем мы сделаем это, давайте рассмотрим другие поля в выводе команды ps. Поле TTY — это сокращение от *teletype* (телетайп), оно содержит информацию об управляющем терминале процесса. В Unix в этом поле выводится тип терминала. Поле TIME содержит объем процессорного времени, потребленного процессом. Как видите, ни один из процессов не является слишком обременительным для компьютера.

Если добавить параметр x, можно получить более богатую информацию о происходящем в системе:

[me@linuxbox ~]$ **ps x**

  PID TTY      STAT   TIME COMMAND

2799 ?        Ssl    0:00 /usr/libexec/bonobo-activation-server –ac

2820 ?        Sl     0:01 /usr/libexec/evolution-data-server-1.10 --

15647 ?        Ss     0:00 /bin/sh /usr/bin/startkde

15751 ?        Ss     0:00 /usr/bin/ssh-agent /usr/bin/dbus-launch --

15754 ?        S      0:00 /usr/bin/dbus-launch --exit-with-session

15755 ?        Ss     0:01 /bin/dbus-daemon --fork --print-pid 4 –pr

15774 ?        Ss     0:02 /usr/bin/gpg-agent -s –daemon

15793 ?        S      0:00 start\_kdeinit --new-startup +kcminit\_start

15794 ?        Ss     0:00 kdeinit Running...

15797 ?        S      0:00 dcopserver –nosid

и еще много других процессов...

Дополнительный параметр x (обратите внимание на отсутствие дефиса) сообщает команде ps, что та должна вывести все процессы, независимо от того, какие терминалы (если таковые имеются) управляют ими. Символ ? в поле TTY указывает на отсутствие управляющего терминала. Таким образом, параметр x позволяет увидеть все процессы в системе, которыми мы владеем.

Так как в системе одновременно выполняется множество процессов, ps производит довольно длинные списки. Часто бывает полезно передать вывод ps команде less через конвейер, чтобы его проще было просматривать. Некоторые комбинации параметров приводят к выводу очень длинных строк, поэтому нелишним будет также распахнуть окно эмулятора терминала на весь экран.

В этом примере в выводе появился новый столбец — STAT. Название STAT — это сокращение от *state* (состояние), столбец содержит информацию о текущем состоянии процесса, как показано в табл. 10.1.

**Таблица 10.1. Состояния процессов**

|  |  |
| --- | --- |
| **Состояние** | **Значение** |
| R | Выполняется. Процесс выполняется или готов к выполнению |
| S | Приостановлен. Процесс временно не выполняется; скорее всего, находится в ожидании определенного события, такого как нажатие клавиши или прибытие сетевого пакета |
| D | Приостановлен без возможности прерывания. Процесс ожидает завершения операции ввода/вывода, например, дисковым устройством |
| T | Остановлен. Процесс принудительно остановлен (подробнее об этом рассказывается ниже) |
| Z | Недействующий процесс-«зомби». Это дочерний процесс, который завершился, но не был удален родителем |
| < | Высокоприоритетный процесс. Существует возможность наиболее важным процессам выделить больше процессорного времени. Данное свойство процесса называется *niceness* (уступчивость). Про процессы с более высокими приоритетами говорят, что они менее уступчивы, потому что потребляют больше процессорного времени, оставляя меньше другим процессам |
| N | Низкоприоритетный процесс. Процесс с низким приоритетом (или уступчивый процесс) получает процессорное время только после того, как будут обслужены процессы с более высоким приоритетом |

Символ, описывающий состояние процесса, может сопровождаться другими символами. Они отражают некоторые экзотические характеристики процессов. За дополнительной информацией обращайтесь к странице справочного руководства (man) для ps.

Еще одна популярная комбинация параметров — aux (без дефиса в начале). Она позволяет получить еще больше информации:

[me@linuxbox ~]$ ps aux

USER    PID %CPU %MEM    VSZ   RSS TTY     STAT START    TIME COMMAND

root      1  0.0  0.0   2136   644 ?       Ss   Mar05    0:31 init

root      2  0.0  0.0      0     0 ?       S<   Mar05    0:00 [kt]

root      3  0.0  0.0      0     0 ?       S<   Mar05    0:00 [mi]

root      4  0.0  0.0      0     0 ?       S<   Mar05    0:00 [ks]

root      5  0.0  0.0      0     0 ?       S<   Mar05    0:06 [wa]

root      6  0.0  0.0      0     0 ?       S<   Mar05    0:36 [ev]

root      7  0.0  0.0      0     0 ?       S<   Mar05    0:00 [kh]

и еще много других процессов...

Эта комбинация параметров выводит процессы, принадлежащие всем пользователям. При использовании параметров без начального дефиса команда действует «в стиле BSD». Linux-версия команды ps может имитировать поведение программы ps, используемой в некоторых реализациях Unix. С помощью этих параметров мы получили дополнительные столбцы, описанные в табл. 10.2.

**Таблица 10.2. Заголовки столбцов при выполнении ps в стиле BSD**

|  |  |
| --- | --- |
| **Заголовок** | **Значение** |
| USER | Идентификатор пользователя. Это владелец процесса |
| %CPU | Использование процессора в процентах |
| %MEM | Использование памяти в процентах |
| VSZ | Объем виртуальной памяти |
| RSS | Размер страниц памяти. Объем физической памяти (ОЗУ), используемой процессом, кб |
| START | Время запуска процесса. Для значений свыше 24 часов выводится дата |

**Просмотр состояния процессов в динамике с помощью top**

Команда ps предоставляет массу информации о том, что делается в компьютере, но она дает только мгновенный снимок, то есть возвращаемая ею информация действительна лишь на момент вызова команды. Чтобы увидеть работу компьютера в динамике, воспользуемся командой top:

[me@linuxbox ~]$ **top**

Программа top постоянно обновляет информацию о процессах (по умолчанию с периодом, равным 3 секундам), чтобы показать их активность с течением времени. Имя программы top отражает тот факт, что она используется для просмотра «топа» (наиболее активных) процессов в системе. Вывод команды top делится на две части: сводная информация о системе и таблица процессов, отсортированных по потреблению ими процессора:

top - 14:59:20 up 6:30,  2 users,  load average: 0.07, 0.02, 0.00

Tasks: 109 total,  1 running, 106 sleeping,   0 stopped,   2 zombie

Cpu(s):  0.7%us, 1.0%sy,  0.0%ni, 98.3%id,  0.0%wa,  0.0%hi,  0.0%si

Mem:    319496k total,  314860k used,     4636k free,    19392k buff

Swap:   875500k total,  149128k used,   726372k free,   114676k cach

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM   TIME+  COMMAND

6244 me        39  19 31752 3124 2188 S  6.3  1.0 16:24.42 trackerd

11071 me        20   0  2304 1092  840 R  1.3  0.3  0:00.14 top

6180 me        20   0  2700 1100  772 S  0.7  0.3  0:03.66 dbus-dae

6321 me        20   0 20944 7248 6560 S  0.7  2.3  2:51.38 multiloa

4955 root      20   0  104m 9668 5776 S  0.3  3.0  2:19.39 Xorg

    1 root      20   0  2976  528  476 S  0.0  0.2  0:03.14 init

    2 root      15  -5     0    0    0 S  0.0  0.0  0:00.00 kthreadd

    3 root      RT  -5     0    0    0 S  0.0  0.0  0:00.00 migratio

    4 root      15  -5     0    0    0 S  0.0  0.0  0:00.72 ksoftirq

    5 root      RT  -5     0    0    0 S  0.0  0.0  0:00.04 watchdog

    6 root      15  -5     0    0    0 S  0.0  0.0  0:00.42 events/0

    7 root      15  -5     0    0    0 S  0.0  0.0  0:00.06 khelper

   41 root      15  -5     0    0    0 S  0.0  0.0  0:01.08 kblockd/

   67 root      15  -5     0    0    0 S  0.0  0.0  0:00.00 kseriod

  114 root      20   0     0    0    0 S  0.0  0.0  0:01.62 pdflush

  116 root      15  -5     0    0    0 S  0.0  0.0  0:02.44 kswapd0

Раздел со сводной информацией содержит массу интересных сведений. Описание выводимой в этом разделе информации приводится в табл. 10.3.

**Таблица 10.3. Поля в разделе со сводной информацией команды top**

| **Строка** | **Поле** | **Значение** |
| --- | --- | --- |
| 1 | top | Имя программы |
|  | 14:59:20 | Текущее время |
|  | up 6:30 | Это поле называется *uptime* (продолжительность работы). Показывает время, прошедшее с момента последней загрузки системы. В данном примере система проработала 6½ часа |
|  | 2 users | В системе работают два пользователя |
|  | load average: | Средняя нагрузка (load average) — это число процессов, ожидающих возобновления работы; то есть число процессов, находящихся в состоянии «выполняется» и совместно использующих процессор. Здесь показаны три значения для разных интервалов времени. Первое значение отражает среднюю нагрузку за последние 60 секунд, второе  — за последние 5 минут и третье — за последние 15 минут. Значения ниже 1.0 сообщают, что система не нагружена |
| 2 | Tasks: | Суммарное число процессов в разных состояниях |
|  | 0.7%us | 0,7% процессорного времени затрачено на выполнение *пользовательских* (user) процессов. Под этим подразумеваются процессы за пределами самого ядра |
|  | 1.0%sy | 1,0% процессорного времени затрачено на выполнение *системных* (system) процессов (ядра) |
|  | 0.0%ni | 0,0% процессорного времени затрачено на выполнение уступчивых (nice), то есть низкоприоритетных, процессов |
|  | 98.3%id | 1,0% процессорного времени составили простои |
| 4 | Mem: | Объем использованной физической памяти (ОЗУ) |
| 5 | Swap: | Объем использованного пространства в файле подкачки (виртуальная память) |

Программа top принимает ряд команд с клавиатуры. Наибольший интерес представляет команда h, которая выводит экран со справочной информацией, и q, которая завершает top.

Оба основных окружения рабочего стола включают приложения с графическим интерфейсом, отображающие аналогичную информацию (подобно тому, как это делает **Task Manager** (Диспетчер задач) в Windows), но я считаю, что top лучше своих аналогов с графическим интерфейсом, потому что она работает быстрее и потребляет меньше системных ресурсов. В конце концов, программа мониторинга системы не должна замедлять систему, за которой мы наблюдаем.

**Управление процессами**

Теперь, когда мы можем видеть процессы и наблюдать за ними, можно приступать к управлению ими. Роль подопытной морской свинки в наших экспериментах исполнит маленькая программка xlogo. Программа xlogo — это демонстрационная программа, поставляемая в составе X Window System (механизм создания графического изображения на дисплее), которая просто отображает окно с логотипом X. Для начала давайте познакомимся с объектом экспериментов:

[me@linuxbox ~]$ **xlogo**

После ввода команды на экране должно появиться небольшое окно с логотипом. В некоторых системах xlogo может выводить предупреждающее сообщение, но его можно смело игнорировать.

**ПРИМЕЧАНИЕ**

Если программа xlogo отсутствует в системе, попробуйте вместо нее использовать gedit или kwrite.

Чтобы убедиться, что xlogo работает, попробуйте изменить размеры ее окна. Если после изменения размеров содержимое окна перерисовывается, значит, программа работает.

Заметили ли вы, что командная оболочка не вывела приглашения к вводу после выполнения команды? Это объясняется тем, что командная оболочка ждет, пока программа завершится. То же самое происходило со всеми программами, которые мы запускали до сих пор. Если закрыть окно xlogo, оболочка выведет приглашение к вводу.

**Прерывание процесса**

Давайте понаблюдаем, что происходит после запуска xlogo. Сначала введите коман­ду xlogo и убедитесь, что программа работает. Затем вернитесь в окно терминала и нажмите комбинацию **CTRL+C**.

[me@linuxbox ~]$ **xlogo**

[me@linuxbox ~]$

Комбинация **CTRL+C** в терминале прерывает выполнение программы. Фактически мы вежливо попросили программу завершиться. После нажатия **CTRL+C** окно xlogo закроется и командная оболочка выведет приглашение к вводу.

Таким способом можно прервать выполнение многих (но не всех) программ команд­ной строки.

**Перевод процессов в фоновый режим**

Представьте, что нам потребовалось вернуться в командную оболочку, не прерывая выполнения программы xlogo. Мы можем сделать это, переведя программу в фоновый режим работы. Считайте, что терминал имеет *передний план* (то, что видно на поверхности, например приглашение к вводу) и *задний план* (*фон*, то, что скрыто под поверхностью). Чтобы запустить программу сразу в фоновом режиме, нужно добавить в конец команды символ амперсанда (&):

[me@linuxbox ~]$ **xlogo &**

[1] 28236

[me@linuxbox ~]$

После ввода такой команды на экране появится окно xlogo, а командная оболочка вернется в приглашение к вводу, но перед этим выведет таинственные числа. Это сообщение является частью механизма *управления заданиями* (job control). Таким способом командная оболочка сообщает, что мы запустили задание с номером 1 ([1]) и оно получило идентификатор процесса PID 28236. Если теперь выполнить команду ps, можно увидеть этот процесс:

[me@linuxbox ~]$ **ps**

  PID TTY          TIME CMD

10603 pts/1    00:00:00 bash

28236 pts/1    00:00:00 xlogo

28239 pts/1    00:00:00 ps

Механизм управления заданиями также дает возможность вывести список заданий, запущенных в терминале. Этот список можно получить командой jobs:

[me@linuxbox ~]$ **jobs**

[1]+ Running                 xlogo &

Результаты показывают, что у нас имеется одно выполняющееся задание с номером 1, которое было запущено командой xlogo &.

**Возврат процесса на передний план**

Процесс в фоновом режиме не получает ввод с клавиатуры, в том числе не видит попыток прервать его комбинацией **CTRL+C**. Вернуть процесс на передний план можно командой fg, как в следующем примере:

[me@linuxbox ~]$ **jobs**

[1]+ Running xlogo &

[me@linuxbox ~]$ **fg %1**

xlogo

За командой fg должен следовать знак процента и номер задания (эта комбинация называется *спецификатором задания*, или *jobspec*). Если имеется только одно фоновое задание, спецификатор можно опустить. Теперь завершим xlogo вводом **CTRL+C**.

**Приостановка процесса**

Иногда необходимо приостановить процесс на время, не завершая его. Это часто делается с целью перевести процесс переднего плана в фоновый режим. Чтобы приостановить процесс переднего плана, используйте комбинацию **CTRL+Z**. Давайте попробуем. В командной строке введите команду xlogo, нажмите **ENTER**, а затем комбинацию **CTRL+Z**:

[me@linuxbox ~]$ **xlogo**

[1]+ Stopped xlogo

[me@linuxbox ~]$

После приостановки xlogo убедимся, что программа действительно приостановилась, для этого попытаемся изменить размер окна xlogo. Увы, программа никак не реагирует на наши действия. Далее можно или вернуть программу на передний план командой fg, или перевести ее в фоновый режим командой bg:

[me@linuxbox ~]$ **bg %1**

[1]+ xlogo &

[me@linuxbox ~]$

Так же как в случае с командой fg, спецификатор задания можно опустить, если имеется только одно задание.

Возможность перевода в фоновый режим полезна и в том случае, если при запуске программы с графическим интерфейсом из командной строки вы забыли добавить в конец команды символ &.

Зачем может понадобиться запускать программу с графическим интерфейсом из командной строки? Тому есть две причины. Во-первых, необходимая программа может отсутствовать в меню программ окружения рабочего стола (как, например, xlogo).

Во-вторых, запуская программу из командной строки, можно увидеть сообщения об ошибках, которые невидимы, когда программа запускается из графического интерфейса. Иногда программа аварийно завершается при запуске из графического меню. В этом случае, запуская ее из командной строки, можно по сообщениям об ошибках понять причину аварии. Кроме того, некоторые программы с графическим интерфейсом имеют интересные параметры командной строки.

**Сигналы**

Команда kill используется для «убийства» (kill), то есть для завершения процессов. Она позволяет принудительно завершить выполнение вышедшей из-под контроля программы, отвергающей любые другие попытки закрыть ее. Например:

[me@linuxbox ~]$ **xlogo &**

[1] 28401

[me@linuxbox ~]$ **kill 28401**

[1]+ Terminated xlogo

В этом случае сначала выполняется запуск программы xlogo в фоновом режиме. В ответ командная оболочка выводит номер задания и идентификатор фонового процесса (PID). Далее вызывается команда kill, которой передается PID процесса, требующего завершения. Процесс можно также идентифицировать, указав спецификатор задания (например, %1) вместо PID.

Хотя все выглядит достаточно просто, в действительности команда kill не просто «убивает» (kill) процессы — она посылает им *сигналы*. Сигналы — один из нескольких способов, которыми операционная система общается с программами. Мы уже видели сигналы в действии на примере использования комбинаций клавиш **CTRL+C** и **CTRL+Z**. Когда терминал принимает одну из этих комбинаций, он посылает сигнал программе на переднем плане. В случае нажатия **CTRL+C** программе посылается сигнал INT (Interrupt — прервать); в случае нажатия **CTRL+Z** посылается сигнал TSTP (Terminal Stop — сигнал «стоп» с клавиатуры). Программы в свою очередь, принимают сигналы и могут реагировать на них. Эта возможность позволяет программе выполнить некоторые операции, например сохранить промежуточные результаты, при получении сигнала на завершение.

**Посылка сигналов процессам командой kill**

Наиболее типичный синтаксис команды kill имеет следующий вид:

kill [-*сигнал*] PID...

Если *сигнал* явно не указан в команде, по умолчанию посылается сигнал TERM (terminate — завершить). Команда kill чаще всего используется для посылки сигналов, перечисленных в табл. 10.4.

**Таблица 10.4. Часто используемые сигналы**

| **Номер** | **Имя** | **Значение** |
| --- | --- | --- |
| 1 | HUP | Обрыв связи. Это пережиток старых добрых времен, когда терминалы подключались к удаленным компьютерам посредством телефонных линий и модемов. Этот сигнал используется, чтобы подсказать программе, что потеряна связь с управляющим терминалом. Действие этого сигнала можно продемонстрировать, закрыв окно терминала. Программа переднего плана, запущенная в терминале, получит сигнал и завершится.  Этот сигнал также используется многими программами-демонами для повторной инициализации. То есть когда программа-демон получает этот сигнал, она перезапускается и повторно читает свои конфигурационные файлы. Веб-сервер Apache, например, как раз такая программа-демон, она именно так реагирует на сигнал HUP |
| 2 | INT | Прервать. Выполняет ту же функцию, что и нажатие комбинации **CTRL+C** в терминале. Обычно приводит к завершению программы |
| 9 | KILL | Уничтожить. Это специальный сигнал. В большинстве случаев программы могут сами решать, как реагировать на сигналы, вплоть до полного их игнорирования, но сигнал KILL в действительности никогда не передается целевой программе. Вместо этого ядро немедленно завершает указанный процесс. Когда процесс завершается таким способом, он не имеет возможности «прибрать за собой» или сохранить результаты своей работы. По этой причине сигнал KILL следует использовать только как крайнее средство, когда другие сигналы на завершение программы не приводят к желаемому результату |
| 15 | TERM | Завершить. Это сигнал по умолчанию, посылаемый командой kill. Если программа достаточно «живая», чтобы принять этот сигнал, она завершится |
| 18 | CONT | Продолжить. Этот сигнал восстанавливает нормальную работу процесса после сигнала STOP |
| 19 | STOP | Приостановить. Этот сигнал заставляет процесс приостановиться, не завершаясь. Подобно сигналу KILL, он не передается целевому процессу и потому не может быть проигнорирован им |

Поэкспериментируем с командой kill:

[me@linuxbox ~]$ **xlogo &**

[1] 13546

[me@linuxbox ~]$ **kill -1 13546**

[1]+ Hangup xlogo

Здесь мы запустили программу xlogo в фоновом режиме и затем с помощью коман­ды kill послали ей сигнал HUP. Программа xlogo завершилась, и командная оболочка сообщила, что фоновый процесс принял сигнал разрыва связи. Иногда необходимо нажать клавишу **ENTER** пару раз, чтобы увидеть сообщение. Обратите внимание, что сигнал можно указать по номеру или по имени, включая имена сигналов, начинающиеся с префикса SIG:

[me@linuxbox ~]$ **xlogo &**

[1] 13601

[me@linuxbox ~]$ **kill -INT 13601**

[1]+ Interrupt xlogo

[me@linuxbox ~]$ **xlogo &**

[1] 13608

[me@linuxbox ~]$ **kill -SIGINT 13608**

[1]+ Interrupt xlogo

Повторите пример, приведенный выше, и попробуйте послать другие сигналы. Имейте в виду, что вместо PID можно также передавать спецификатор задания.

Процессы, подобно файлам, имеют владельцев, и чтобы послать сигнал процессу командой kill, вы должны быть владельцем процесса (или суперпользователем).

Помимо сигналов, наиболее часто используемых с командой kill и перечисленных в табл. 10.4, система часто использует другие сигналы, перечисленные в табл. 10.5.

**Таблица 10.5. Другие часто используемые сигналы**

|  |  |  |
| --- | --- | --- |
| **Номер** | **Имя** | **Значение** |
| 3 | QUIT | Выйти |
| 11 | SEGV | Ошибка сегментации. Этот сигнал посылается программе, предпринявшей попытку недопустимого обращения к памяти, то есть попытку выполнить запись в память, доступ к которой запрещен |
| 20 | TSTP | Сигнал «стоп» с клавиатуры. Этот сигнал посылается терминалом после нажатия комбинации **CTRL+Z**. В отличие от сигнала STOP, TSTP передается программе, и программа может решить игнорировать его |
| 28 | WINCH | Изменение окна. Этот сигнал посылается системой при изменении размеров окна терминала. Некоторые программы, такие как top и less, реагируют на этот сигнал, обновляя свой вывод в соответствии с новыми размерами окна терминала |

Любопытные пользователи могут получить полный список сигналов, выполнив следующую команду:

[me@linuxbox ~]$ **kill -l**

**Посылка сигналов нескольким процессам с помощью killall**

Кроме того, существует возможность с помощью команды killall послать сигнал сразу нескольким процессам, соответствующим указанной программе или имени пользователя. Она имеет следующий синтаксис:

killall [-u *пользователь*] [-*сигнал*] *имя*...

Для демонстрации запустим пару экземпляров программы xlogo и затем завершим их:

[me@linuxbox ~]$ **xlogo &**

[1] 18801

[me@linuxbox ~]$ **xlogo &**

[2] 18802

[me@linuxbox ~]$ **killall xlogo**

[1]- Terminated xlogo

[2]+ Terminated xlogo

Помните: так же как при использовании команды kill, вы должны обладать привилегиями суперпользователя, чтобы посылать сигналы процессам, которыми не владеете.

**Другие команды управления процессами**

Так как мониторинг процессов является одной из важнейших задач системного администрирования, существует множество команд, помогающих в этом. В табл. 10.6 перечислены некоторые из них, с ними вы можете поэкспериментировать.

**Таблица 10.6. Другие команды управления процессами**

|  |  |
| --- | --- |
| **Команда** | **Описание** |
| pstree | Выводит список процессов в виде древовидной структуры, отражающей отношения «родитель—потомок» между процессами |
| vmstat | Выводит мгновенный снимок с информацией об использовании системных ресурсов, включая память, файл подкачки и объем дискового ввода/вывода. Чтобы увидеть, как изменяется эта информация с течением времени, передайте команде интервал задержки (в секундах) между обновлениями (например, vmstat 5). Завершить работу команды можно нажатием **CTRL+C** |
| xload | Программа с графическим интерфейсом, показывающая изменение нагрузки на систему с течением времени |
| tload | Работает подобно программе xload, но рисует график в терминале. Завершается работа команды нажатием **CTRL+C** |