

Общая версия Linux. Уровень 1

Загрузка ОС и процессы



Оглавление

[Загрузка операционной системы](#)

[Процесс. Управление процессами](#)

[Атрибуты процессов](#)

[Управление процессами](#)

[Классификация сигналов](#)

[Стандартные потоки. Конвейер \(pipeline\)](#)

[Мониторинг процессов и состояния компьютера](#)

[Практическое задание](#)

[Дополнительные материалы](#)

[Используемые источники](#)

На этом уроке

1. Узнаем, как происходит процесс загрузки ОС.
2. Познакомимся с терминалом — основным интерфейсом взаимодействия ОС и пользователя.
3. Выясним, что такое процесс и как управлять процессами.
4. Выясним, как перенаправить результат работы одной команды на ввод другой, используя конвейер (pipeline).
5. Разберём простейшие утилиты мониторинга процессов.

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. Systemd — это надстройка системы инициализации System V. 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.

Потоки — программный интерфейс для чтения или записи файлов, организации межпроцессорного взаимодействия.

Стандартные потоки — специальный тип потоков, имеющих свой номер (дескриптор) и предназначенных для выполнения стандартных функций: ожидание команд пользователя, вывод данных на экран, вывод сообщений об ошибках на экран.

Конвейер (pipeline) — набор процессов, для которых реализована следующая схема: вывод результата работы одного процесса передаётся на ввод другому процессу.

Загрузка операционной системы

Загрузка операционной системы после включения компьютера происходит в несколько шагов:

1. Запуск BIOS/UEFI. На этом этапе происходит диагностика оборудования и запускается загрузчик операционной системы.
2. Запуск загрузчика. На этом этапе происходит запуск ядра операционной системы. Стандартный загрузчик в ОС Linux — [GRUB](#) (сокр. от англ. **GR**and **U**nified **B**ootloader — «основной единый загрузчик»). Он позволяет загрузить любую совместимую с ним систему (Windows, Linux, Unix), выбрать ядро, с которым будет работать операционная система, или передать параметры ядру операционной системы, например для загрузки в режим восстановления.
3. Запуск ядра. На этом этапе происходит запуск ядра с выбранными установками и монтирование виртуальной файловой системы `initrd`.
4. Запуск процесса инициализации системы. На этом этапе ядро запускает самый первый процесс `systemd` (в более старых версиях процесс назывался `init`), который запускает все остальные процессы, необходимые для корректной работы операционной системы.
5. Завершается загрузка операционной системы либо запуском графической подсистемы (если устанавливали десктоп-версию Ubuntu), либо запуском терминала. Терминал — основное средство взаимодействия между пользователем и операционной системой. Через терминал пользователь вводит команды, и на терминал операционная система выводит результат работы команд.

Процесс. Управление процессами

Процесс — одно из основополагающих понятий в ОС Linux. По сути, это совокупность какого-то кода, выполняющегося в памяти компьютера, но есть приложения, которые могут создавать в результате

своей работы не один, а несколько процессов. Каждая команда, которую мы выполняем в терминале, или приложение, которое мы запускаем в графической оболочке также порождает процессы.

Процесс может находиться в следующих состояниях:

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 «зомби» — процесс уже не существует, его код и данные уже выгружены из оперативной памяти компьютера, но запись в таблице процессов по какой-то причине сохранилась.

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

Атрибуты процессов

Как видно из определения таблицы 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 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**. При завершении работы родительского процесса процесс-«потомок» также прекратит свою работу, так как его родительский процесс отправит команду на завершение работы всех своих «потомков».
2. **UID** — владелец процесса, пользователь, от которого запущен процесс.
3. **CMD** — команда, запустившая процесс.

Список запущенных процессов и их атрибутов позволит увидеть команда **ps -ef**. Параметры **-ef** выведут на экран все процессы со всеми атрибутами.

Управление процессами

[Процессы в Linux](#) можно разделить на три типа:

1. **Системные процессы.** Они не имеют собственных исполняемых файлов и запускаются из ядра операционной системы. Стартуют при запуске операционной системы и существуют всё время работы ОС. Пример такого процесса — **systemd**.
2. **Демоны.** Это процессы, которые также не общаются с пользователем напрямую. Управление ими осуществляется через утилиту **systemctl**. Они выполняются в фоновом режиме и отвечают за предоставление сервисов. Например, служба **apache2** осуществляет

предоставление веб-ресурсов. Каждому демону соответствует исполняемый файл, который обычно запускается автоматически при загрузке системы.

3. **Пользовательские процессы.** Запускаются из исполняемого (бинарного) файла пользователем. Они могут выполняться в интерактивном или фоновом режиме. Срок их выполнения ограничен продолжительностью сеанса работы пользователя. Любая выполненная команда или запущенная программа — это пользовательский процесс.

Управление системными процессами и процессами-демонами осуществляется через утилиту **systemctl**. **Systemctl** — основная команда для управления и мониторинга **systemd**, позволяет получать информацию о состоянии системы и запущенных службах, а также управлять службами. Более подробную информацию можно получить на страницах справочного руководства **man systemctl**.

Основные параметры **systemctl**:

1. `Systemctl status` выведет на экран состояние системы.
2. `Systemctl` выведет список запущенных юнитов. С точки зрения `systemctl` юнитом может быть служба, точка монтирования дискового устройства.
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`).
4. `Systemctl [enable|disable] service_name` позволит добавить (`enable`) или убрать (`disable`) службу из автозагрузки.

Пользовательские процессы, как правило, живут до окончания пользовательской сессии или обрыва терминала, но бывают случаи, когда какое-то приложение или запущенный процесс необходимо завершить аварийно. Такие ситуации возникают не только с пользовательскими процессами, но и с процессами-демонами. Для этого существуют специальные сигналы, которые мы можем передать процессу, используя команду **kill**. Полный список сигналов можно получить, выполнив команду **kill -l**. Команда **kill** работает с процессом через его **PID** или **PPID**. Передав сигнал на принудительное завершение **PPID** (ID родительского процесса), мы завершим всё дерево процессов, порождённых этим процессом.

Классификация сигналов

Существует [28 сигналов](#), которые можно классифицировать следующим образом:

Название	Код	Действие по умолчанию	Описание	Тип
SIGABRT	6	Завершение с дампом памяти	Сигнал, посылаемый функцией <code>abort()</code>	Управление

SIGALRM	14	Завершение	Сигнал истечения времени, заданного alarm()	Уведомление
SIGBUS	10	Завершение с дампом памяти	Неправильное обращение в физическую память	Исключение
SIGCHLD	18	Игнорируется	Дочерний процесс завершён или остановлен	Уведомление
SIGCONT	25	Продолжить выполнение	Продолжить выполнение ранее остановленного процесса	Управление
SIGFPE	8	Завершение с дампом памяти	Ошибочная арифметическая операция	Исключение
SIGHUP	1	Завершение	Закрытие терминала	Уведомление
SIGILL	4	Завершение с дампом памяти	Недопустимая инструкция процессора	Исключение
SIGINT	2	Завершение	Сигнал прерывания (Ctrl-C) с терминала	Управление
SIGKILL	9	Завершение	Безусловное завершение	Управление
SIGPIPE	13	Завершение	Запись в разорванное соединение (пайп, сокет)	Уведомление
SIGQUIT	3	Завершение с дампом памяти	Сигнал Quit с терминала (Ctrl-\)	Управление
SIGSEGV	11	Завершение с дампом памяти	Нарушение при обращении в память	Исключение
SIGSTOP	23	Остановка процесса	Остановка выполнения процесса	Управление

SIGTERM	15	Завершение	Сигнал завершения (сигнал по умолчанию для утилиты kill)	Управление
SIGTSTP	20	Остановка процесса	Сигнал остановки с терминала (Ctrl-Z).	Управление
SIGTTIN	26	Остановка процесса	Попытка чтения с терминала фоновым процессом	Управление
SIGTTOU	27	Остановка процесса	Попытка записи на терминал фоновым процессом	Управление
SIGUSR1	16	Завершение	Пользовательский сигнал №1	Пользовательский
SIGUSR2	17	Завершение	Пользовательский сигнал №2	Пользовательский
SIGPOLL	22	Завершение	Событие, отслеживаемое poll()	Уведомление
SIGPROF	29	Завершение	Истечение таймера профилирования	Отладка
SIGSYS	12	Завершение с дампом памяти	Неправильный системный вызов	Исключение
SIGTRAP	5	Завершение с дампом памяти	Ловушка трассировки или breakpoint	Отладка
SIGURG	21	Игнорируется	На соquete получены срочные данные	Уведомление
SIGVTALRM	28	Завершение	Истечение «виртуального таймера»	Уведомление
SIGXCPU	30	Завершение с дампом памяти	Процесс превысил лимит процессорного времени	Исключение
SIGXFSZ	31	Завершение с дампом памяти	Процесс превысил допустимый размер файла	Исключение

Стандартные потоки. Конвейер (pipeline)

Каждому процессу в Linux сопоставляется таблица открытых им файлов, которая располагается в адресном пространстве ядра. Первые три позиции в этой таблице всегда отдаются под три специальных файла, посредством которых процесс осуществляет взаимодействие с пользователем (получает информацию, выводит информацию на экран, сообщает об ошибках во время работы). Эти три файла называются стандартными потоками, и ссылаться на эти файлы можно, используя их дескрипторы. Файловый дескриптор — целое число, соответствующее открытому файлу:

0 — стандартный поток ввода (**STDIN**). Файл, из которого осуществляется чтение данных.

1 — стандартный поток вывода (**STDOUT**). Файл, в который осуществляется запись данных.

2 — стандартный поток ошибок (**STDERR**). Файл, в который осуществляется запись об ошибках или сообщения, которые не могут быть записаны в стандартный поток вывода.

Информацию из потоков можно перенаправить в любой другой файл, используя следующие символы:

1. **<** — перенаправление стандартного ввода, например, **<file**. Процесс будет использовать файл **file** как источник данных.
2. **>**, **>>** — перенаправление стандартного вывода, например, **>file**. Сообщения от процесса будут направлены в файл с именем **file**. Если файл не существует, он будет создан, в противном случае перезаписан. Для записи сообщений в конец файла используется символ **>>**, например, **>>file**. Сообщения от процесса будут перенаправлены в файл с именем **file**. Если файл не существует, он будет создан. В противном случае сообщения будут дописаны в конец файла.
3. **2>**, **2>>** — перенаправление стандартного потока ошибок, например, **2>file**. Сообщения об ошибках процесса будут направлены в файл с именем **file**. Если файл не существует, он будет создан, в противном случае перезаписан. Для записи сообщений в конец файла используется символ **2>>**, например, **2>>file**. Сообщения об ошибках процесса будут перенаправлены в файл с именем **file**. Если файл не существует, он будет создан, в противном случае сообщения будут дописаны в конец файла.
4. **2>&1** — позволит объединить поток ошибок (**stderr**) и стандартный вывод (**stdout**) и перенаправить данные в другой файл.

Стандартные потоки можно перенаправлять не только в файлы, но и на ввод другим процессам. Такое перенаправление называют конвейер (pipeline), в нём используется специальный символ «**|**» (вертикальная черта). Например, `command-1 | command-2 | ... | command-n` перенаправит результат работы команды `command-1` на ввод другой команде — `command-2`, которая, в свою очередь, перенаправит результат своей работы на ввод следующей команде. Такое перенаправление очень

часто используется в работе с командной строкой Linux. Например, используя такое перенаправление, мы можем подсчитать количество объектов в текущем каталоге.

`ls |wc -l` — команда `ls` выполнит листинг содержимого и вместо вывода данных на экран отправит её на ввод команде `wc -l`, которая подсчитывает количество строк в листинге.

Мониторинг процессов и состояния компьютера

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

1. **Ps** покажет список запущенных процессов в операционной системе. Используя её в сочетании с командой **grep** (утилита, осуществляющая поиск по строкам согласно заданному шаблону), мы можем найти и получить следующую информацию о процессе: PID, PPID, статус процесса.
2. **Top** (table of process) выведет список и информацию о запущенных в системе процессах, покажет общую загрузку системы — строка `load average`. При этом важно понимать, что значения `load average` большие, чем количество доступных ядер процессора, говорят о высокой нагрузке на сервер. Также программа по умолчанию сортирует процессы по нагрузке на процессор в режиме реального времени.
3. **Df** покажет размер, занятое и свободное пространство на смонтированных файловых системах в ОС.

Также полезно проводить анализ работы ОС и запущенных в ней служб, используя информацию из файлов логов, которые расположены в каталоге `/var/log`, либо используя утилиту **journalctl**. Например, `journalctl -u sshd` покажет журнал работы процесса `sshd`.

Практическое задание

1. **Потоки ввода/вывода.** Создать файл, используя команду `echo`. Используя команду `cat`, прочитать содержимое каталога `etc`, ошибки перенаправить в отдельный файл.
2. **Конвейер (pipeline).** Использовать команду `cut` на вывод длинного списка каталога, чтобы отобразить только права доступа к файлам. Затем отправить в конвейере этот вывод на `sort` и `uniq`, чтобы отфильтровать все повторяющиеся строки.
3. **Управление процессами.** 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 (services sshd reload). В чём различие между действиями restart и reload? Создайте файл при помощи команды `cat > file_name`, напишите текст и завершите комбинацией ctrl+d. Какой сигнал передадим процессу?

4. **Сигналы процессам.** Запустите `mc`. Используя `ps`, найдите PID процесса, завершите процесс, передав ему сигнал 9.

Дополнительные материалы

[Потоки ввода/вывода](#)

[Процессы в Linux](#)

Используемые источники

[Загрузка операционной системы](#)

[Процессы в Linux](#)

[Робачевский А. Операционная система Unix](#)

[Костромин В. Linux для пользователя](#)