8 Управление процессами

8.2

Любой процесс обладает уникальным **идентификатором процесса**, который представляет собой целое положительное число. Поскольку идентификатор процесса — это единственный широко используемый идентификатор**, уникальность** которого **гарантируется системой**, он часто присоединяется к другим идентификаторам для придания им уникальности.

По завершении процесса его идентификатор может использоваться повторно для другого процесса.

**Процесс с** **идентификатором 0** — это, как правило, процесс-планировщик, который часто называют swapper (**программа подкачки**). Этому процессу не соответствует никакая программа на диске, поскольку он является частью ядра и считается системным процессом.

**Процесс с идентификатором 1** — это обычно процесс init, который запускается ядром в конце процедуры начальной загрузки. Этот процесс отвечает за **запуск операционной системы** после загрузки ядра. Процесс init никогда не «умирает». Это обычный пользовательский процесс, он не является системным процессом ядра, как swapper, хотя и обладает привилегиями суперпользователя

8.11

Когда программе необходимы **дополнительные привилег**ии, чтобы получить доступ к ресурсам, недоступным в настоящее время, она должна **изменить свой идентификатор пользователя или группы** на идентификатор, который имеет соответствующие привилегии. Точно так же, чтобы понизить свои привилегии или предотвратить доступ к некоторым ресурсам, программа должна изменить идентификатор пользователя или группы на идентификатор, не обладающий указанными привилегиями или достаточными правами для обращения к ресурсу.

4.4

**Реальные идентификаторы пользователя и группы** определяют, кто мы на самом деле. Эти идентификаторы извлекаются из файла паролей во время входа в систему. Обычно в течение сеанса значения этих идентификаторов не меняются, хотя процессы, обладающие правами суперпользователя, имеют возможность изменять их, о чем мы поговорим в разделе 8.11.

**Эффективные идентификаторы пользователя и группы** и идентификаторы дополнительных групп определяют права доступа к файлам, о чем мы поговорим в следующем разделе. (Определение дополнительных групп было дано в разделе 1.8.)

**Сохраненные идентификаторы пользователя и группы** — это копии эффективных идентификаторов, которые создаются в момент запуска программы. Мы расскажем о назначении этих двух идентификаторов, когда будем описывать функцию setuid в разделе 8.11.

9.4

**Группа процессов** — это коллекция из одного или более процессов, обычно связанных с выполнением одного и того же задания (управление заданиями рассматривается в разделе 9.8), которые могут принимать сигналы от одного и того же терминала. Каждая группа процессов имеет уникальный идентификатор.

Каждая группа процессов **может** иметь лидера. **Идентификатор группы процессов лидера группы совпадает с его идентификатором процесса. (gid лидера группы = его id)**

Вполне допустима ситуация, когда лидер группы создает группу процессов, затем запускает процессы в этой группе и завершается. **Группа процессов продолжит существовать, пока в ней остается хотя бы один процесс**, вне зависимости от того, завершил работу лидер группы или нет. Период от момента создания группы и до момента, когда последний процесс в группе покинет ее, называется временем жизни группы процессов. Последний оставшийся в группе процесс может либо завершиться, либо войти в состав другой группы процессов.

Процесс может установить идентификатор группы только для себя самого и для своих дочерних процессов. Но процесс не может изменить идентификатор группы процессов дочернего процесса, который вызвал одну из функций семейства exec.

9.5

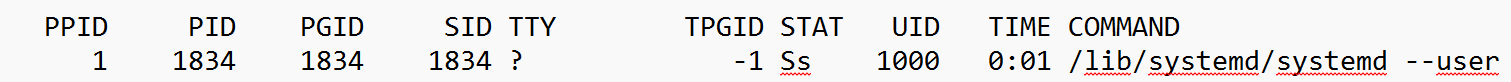
Сеанс — это коллекция из одной или более групп процессов

9.6

Сеансы и группы процессов обладают еще некоторыми характеристиками.

*  **Сеанс может иметь только один управляющий терминал**. Обычно это устройство терминала (в случае входа в систему с терминала) или устройство псевдотерминала (в случае входа в систему через сетевое соединение), с которого был произведен вход в систему.
*  Лидер сеанса, который устанавливает соединение с управляющим терминалом, называется управляющим процессом.
*  Группы процессов в пределах одного сеанса могут подразделяться на единственную группу процессов переднего плана и одну или более групп фоновых процессов.
*  Если сеанс имеет управляющий терминал, в нем будет одна группа процессов переднего плана, а все остальные группы процессов в сеансе будут группами фоновых процессов.
*  Когда мы вводим с клавиатуры терминала символ прерывания (обычно DELETE или Control-C), всем процессам в группе процессов переднего плана посылается сигнал прерывания.
*  Когда мы вводим с клавиатуры терминала символ завершения (обычно Control-\), всем процессам в группе процессов переднего плана посылается сигнал завершения.

Процесс-посредник для осиротевших процессов



/lib/systemd/systemd - символическая ссылка на /sbin/init

system - linux man

<https://man7.org/linux/man-pages/man1/systemd.1.html>

/usr/lib/systemd/systemd [OPTIONS...]

**systemd is a system and service manager for Linux operating**

**systems.** When run as first process on boot (as PID 1), it acts as

init system that brings up and maintains userspace services.

Separate instances are started for logged-in users to start their

services.

systemd is usually not invoked directly by the user, but is

**installed as the /sbin/init symlink** and **started during early**

**boot**. The user manager instances are started automatically

through the user@.service(5) service.

13.1

**Демоны** — это долгоживущие процессы. Зачастую они запускаются во время загрузки системы и завершают работу вместе с ней. Так как они **не имеют управляющего терминала**, говорят, что они работают в фоновом режиме.

ps -ajx

-a используется для вывода процессов, которыми владеют другие пользователи,

-x — для вывода процессов, не имеющих управляющего терминала

-j — для вывода дополнительных сведений, имеющих отношение к заданиям: идентификатора сеанса, идентификатора группы процессов, управляющего терминала и идентификатора группы процессов терминала

идентификатор родительского процесса (PPID),

идентификатор процесса (PID),

идентификатор группы процессов (PGID),

идентификатор сеанса (SID),

имя управляющего терминала (TTY)

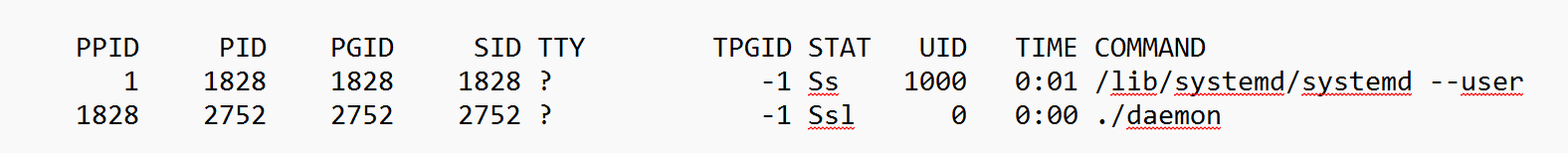
идентификатор группы процессов переднего плана, или -1 (TPGID)

статус процесса (STAT)

идентификатор пользователя (UID)

накопленное процессорное время, пользователь + система (TIME)

строка команды (CMD)



Обратите внимание,

* большинство демонов обладают привилегиями суперпользователя (root).
* ни один из демонов не имеет управляющего терминала
* Все демоны пользовательского уровня являются лидерами групп и лидерами сеансов, а также единственными процессами в своих группах процессов и сеансах (исключение составляет rsyslogd)
* о родителем для большинства демонов является процесс init.

**Правила программирования демонов**

1. Вызвать функцию umask, чтобы сбросить маску режима создания файлов в значение 0. Маска, наследуемая от запускающего процесса, может маскировать некоторые биты прав доступа. Если предполагается, что процесс-демон будет создавать файлы, может потребоваться установка определенных битов прав доступа.

***Режим создания файла*** *— это права доступа, причем права доступа для трех категорий пользователя: пользователя группы и others.*

*Сразу после идет fork(), кот создает процесс потом который наследует: код предка, сигнальную маску, дескрипторы открытых файлов и маску режима создания файлов.*

*Т е на самом деле маску надо сбросить в child.*

1. Вызвать функцию fork и завершить родительский процесс. Зачем?
   1. ***Гарантированно созданный процесс не был лидером группы – это условие вызова функции setsid().*** *Дело в том что когда процесс вызывает fork он становится лидером группы, мы завершаем предка, потомок теряет группу и лидером группы не будет.*
   2. если демон запущен как обычная команда оболочки, завершив родительский процесс, мы заставим командную оболочку думать, что команда выполнилась
   3. дочерний процесс наследует идентификатор группы процессов от родителя, но получает свой идентификатор процесса; тем самым гарантируется, что дочерний процесс не будет являться лидером группы, а это необходимое условие для вызова функции setsid, который будет произведен далее.
2. Создать новый сеанс, обратившись к функции setsid. При этом процесс становится (а) лидером нового сеанса, (б) лидером новой группы процессов и (в) лишается управляющего терминала.

*Установить session id.* ***Лишает процесс управляющего терминала, делает его лидером группы и лидером сессии, но в этой группе и в этой сессии он один.*** *Это видно по выводу ps*

1. Поменять токующую директорию на корневую. Текущий рабочий каталог, унаследованный от родительского процесса, может находиться на смонтированной файловой системе. Поскольку демон, как правило, существует все время, пока система не перезагрузится, в подобной ситуации, когда рабочий каталог демона находится в смонтированной файловой системе, ее невозможно будет отмонтировать.

*Поскольку демоны выполняются продолжительное время, может возникнуть необходимость отмонтировать файловую систему, с которой запущен демон, а он еще нужен в рабочем состоянии. Поэтому предлагают поменять директорию на корневую*

1. Закрыть все ненужные файловые дескрипторы. *Которые унаследовались от родителя*

*получить максимально возможный номер дескриптора файла. – getrlimit() -> struct rlimit*

*rl.rlim\_max = 1024; // макс число файловых дескрипторов которые могут быть открыты*

*Почему такое большое количество файлов может быть открыто одним процессом?*

*Для структурирования данных во времена когда еще не было баз данных.*

***Закрываем все 1024, т е закрываем stdin stdout stderr, а потом котрываем но направляем на /dev/null. У процесса не быть этих потоков***

1. Некоторые демоны открывают файловые дескрипторы с номерами 0, 1 и 2 на устройстве /dev/null, — таким образом, любые библиотечные функции, которые пытаются читать со стандартного устройства ввода или писать в стандартное устройство вывода или сообщений об ошибках, не будут оказывать никакого влияния.

*/dev/null – черная дыра, все что туда сбрасывается – теряется*

1. *Обеспечить невозможность обретения управляющего терминала в будущем.*

*Сигнал SIGHUP будет игнорироваться*

***SIGHUP – разрыв связи с терминалом***

1. *Инициализировать файл журнала.*

13.5 Демоны в единственном экземпляре

Один и тот же демон делает одно и то же, значит обращаться будет к одним и тем же записям. Значит к этим данным надо будет обеспечивать монопольный доступ. А если он в единственном экземпляре, то монопольный доступ реализовывать не надо.

int already\_running(void) – *обеспечивает что в системе демон будет существовать в единственном экземпляре. Для этого используется файл блокировки. Он создается в этой функции и блокируется на чтение и на запись, что видно их кода. Этот файл создается в директории* ***/var/run*** *(чтобы создать файл в этом каталоге, демон должен обладать привилегиями суперпользователя)*

|  |
| --- |
| #define LOCKFILE "/var/run/daemon.pid"  // read\_user, write\_user, read\_group, read\_others  #define LOCKMODE (S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH) |

*Почему для группы и others блокируется только чтение? Предположение: что если ты не можешь читать файл то ты не можешь писать в чужой файл*

Одним из основных механизмов, обеспечивающих ограничение количества одновременно работающих копий демона, являются блокировки файлов и записей. (Блокировки файлов и записей в файлах мы рассмотрим в разделе 14.3.) Если каждый из демонов создаст файл и попытается установить для этого файла блокировку для записи, система разрешит установить только одну такую блокировку. Все последующие попытки установить блокировку для записи будут.

Каждая копия демона будет пытаться создать файл и записать в него свой идентификатор процесса.

* Если файл уже заблокирован, функция **lockfile** завершится неудачей с кодом ошибки EACCESS или EAGAIN в переменной errno и в вызывающую программу вернет значение 1, указывающее, что демон уже запущен.
* Иначе функция усекает размер файла до нуля, записывает в него идентификатор процесса и возвращает значение 0.

Усечение размера файла необходимо, потому что идентификатор процесса предыдущей копии демона, представленный в виде строки, мог иметь большую длину

13.6. Соглашения для демонов

* Если демон использует **файл блокировки**, этот файл помещается в каталог **/var/run**. Однако, чтобы создать файл в этом каталоге, демон должен обладать привилегиями суперпользователя
* Если демон поддерживает определение дополнительных настроек, они обычно хранятся в каталоге **/etc**. Имя **конфигурационного файла**, как правило, имеет вид name.conf, где name — имя демона или службы
* Демоны могут запускаться из командной строки, но чаще запуск демонов производится из сценариев инициализации системы
* Если демон имеет **конфигурационный файл**, настройки из него **читаются** демоном **во время запуска**, и затем он обычно не обращается к этому файлу.

Если в конфигурационный файл были внесены изменения, демон пришлось бы останавливать и перезапускать снова, чтобы новые настройки вступили в силу. Во избежание этого некоторые демоны устанавливают обработчики сигнала **SIGHUP**, в которых производится **чтение конфигурационного файла и перенастройка демона**. Поскольку демоны не имеют управляющего терминала и являются либо лидерами сеансов без управляющего терминала, либо членами осиротевших групп процессов, у них нет причин ожидать сигнала SIGHUP. Поэтому он может использоваться для других целей.

SIGHUP – разрыв связи с терминалом

Далее **блокируются все сигналы**, поскольку это рекомендуется для многопоточных программ, и **создается поток, который будет заниматься обработкой сигналов**. Поток **обслуживает** только **сигналы SIGHUP и SIGTERM**.

При получении сигнала SIGHUP функция reread перечитывает файл конфигурации, а

при получении сигнала SIGTERM поток записывает сообщение в журнал и завершает работу процесса.

В табл. 10.1 указано, что по умолчанию сигналы SIGHUP и SIGTERM завершают процесс. Поскольку эти сигналы заблокированы, демон не будет завершаться, если получит один из них. Вместо этого поток, вызывая sigwait, будет получать номера доставленных сигналов.

*Почему у демона не должно быть управляющего терминала?*

*Чтобы не было возможности влиять на его выполнение*

*Почему демон должен быть в единственном экземпляре?*

*Чтобы не выполнять разделение данных, которое неминуемо если выполняется один и тот же код*

[*https://gaydov.blogspot.com/2013/07/unix-job-control.html*](https://gaydov.blogspot.com/2013/07/unix-job-control.html)

