Документация по API ядра

Бывает так, что компьютер “тормозит” и не позволяет выполнять некоторые важные операции. Такое происходит, когда какой-нибудь процесс занимает большую часть оперативной памяти или процессорного времени. Зная основы работы с процессами можно отключить мешающий процесс и разгрузить компьютер. Существуют ситуации, когда запущенная программа перестает реагировать на действия пользователя и никак не удается ее закрыть штатным способом. В данном случае достаточно найти процесс данной программы и принудительно закрыть программу.

К процессам часто обращаются, когда компьютер содержит вирус или другое вредоносное ПО. Взглянув на процессы можно найти аномальное поведение системы и найти вредоносное ПО. Поэтому даже обычному пользователю необходимо знать основы работы с процессами. Прежде, чем мы рассмотрим как управлять процессами, перечислим основные параметры, характерные для каждого процесса:

PID — (process ID) идентификатор каждого процесса

PPID — (parent process ID) идентификатор родительского процесса. Процесс может порождать и другие процессы.

UID, GID — реальные идентификаторы пользователя и его группы, запустившего данный процесс.

EUID, EGID — эффективные идентификаторы пользователя и его группы. Признаки доступа SUID, SGID. Когда пользователь запускает файл с установленными признаками, то эффективные идентификаторы (EUID, EGID) равны реальным идентификаторам (UID, GID) владельца данного файла. То есть система смотрит на эффективные идентификаторы и таким образом узнает, что был установлен признак SUID/SGID и предоставляет доступ пользователю. Если пользователь запустит любой другой файл без установленного признака SUID/SGID, то реальные и эффективные идентификаторы всегда равны.

Priority/Nice — приоритет и относительный приоритет. Служит для выделения большего или меньшего процессорного времени для определенного процесса. Разрешается менять только относительный приоритет (Nice). Его значение варьируется от -20 до +19. Чем ниже значение относительного приоритета, тем больше процессорного времени выделяется для данного процесса.

STAT — состояние процесса. В таблице представлены обозначения процессов:

Состояние	**Описание**
R (runnable)	Работающий процесс
S (sleeping)	Процесс в состоянии ожидания
T (stopping)	Остановленный процесс
Z (zombie)	Завершившийся процесс

B (uninterruptible)	Непрерывный процесс

Кроме того, помимо состояния самого процесса можно увидеть и дополнительную информацию (индикатор), которая следует сразу за символом состояния:

Индикатор	**Описание**
<	Процесс с высоким приоритетом
N	Процесс с низким приоритетом
L	Процесс имеет страницы, заблокированные в памяти
s	Процесс является лидером сессии
l	
+	Процесс работает на переднем плане

Как же можно увидеть все процессы?

Для этого существует команда *ps*. Однако в “чистом виде” она обычно не используется, поэтому применяются различные опции. Она имеет множество опций, причем некоторые используются с дефисом, а некоторые — без. Поэтому предлагаю просто запомнить наиболее используемые команды.

- Список всех процессов — *ps aux*
- Отображение всех процессов, включая и PPID — *ps -ef*
- Отображение дерева процессов, то есть схематическое отображение от какого процесса был порожден конкретный процесс — *ps auxf*
- Отображение дерева процессов, включая PPID — *ps ajxf*
- Более короткий вывод дерева процессов — *pstree*
- Отображение процессов конкретного пользователя — *ps U пользователь*

Процесс в Linux (как и в UNIX) - это программа, которая выполняется в отдельном виртуальном адресном пространстве. Когда пользователь регистрируется в системе, автоматически создается процесс, в котором выполняется оболочка (shell), например, /bin/bash.

В Linux поддерживается классическая схема мультипрограммирования. Linux поддерживает параллельное (или квазипараллельного при наличии только одного процессора) выполнение процессов пользователя. Каждый процесс выполняется в собственном виртуальном адресном пространстве, т.е. процессы защищены друг от друга и крах одного процесса никак не повлияет на другие выполняющиеся процессы и на всю систему в целом. Один процесс не может прочитать

что-либо из памяти (или записать в нее) другого процесса без "разрешения" на то другого процесса. Санкционированные взаимодействия между процессами допускаются системой.

Ядро предоставляет системные вызовы для создания новых процессов и для управления порожденными процессами. Любая программа может начать выполняться только если другой процесс ее запустит или произойдет какое-то прерывание (например, прерывание внешнего устройства).

В связи с развитием SMP (Symmetric Multiprocessor Architectures) в ядро Linux был внедрен механизм нитей или потоков управления (threads). Нить - это процесс, который выполняется в виртуальной памяти, используемой вместе с другими нитями процесса, который обладает отдельной виртуальной памятью.

Если интерпретатору (shell) встречается команда, соответствующая выполняемому файлу, интерпретатор выполняет ее, начиная с точки входа (entry point). Для С-программ entry point - это функция main. Запущенная программа тоже может создать процесс, т.е. запустить какую-то программу и ее выполнение тоже начнется с функции main.

Для создания процессов используются два системных вызова: fork() и exec. fork() создает новое адресное пространство, которое полностью идентично адресному пространству основного процесса. После выполнения этого системного вызова мы получаем два абсолютно одинаковых процесса - основной и порожденный. Функция fork() возвращает 0 в порожденном процессе и PID (Process ID - идентификатор порожденного процесса) - в основном. PID - это целое число.

Теперь, когда мы уже создали процесс, мы можем запустить программу с помощью вызова exec. Параметрами функции exec является имя выполняемого файла и, если нужно, параметры, которые будут переданы этой программе. В адресное пространство порожденного с помощью fork() процесса будет загружена новая программа и ее выполнение начнется с точки входа (адрес функции main).

Процесс в ядре представляется просто как структура с множеством полей (определение структуры можно прочитать здесь).

Но так как статья посвящена системному программированию, а не разработке ядра, то несколько абстрагируемся и просто акцентируем внимание на важных для нас полях процесса:

Идентификатор процесса (pid)

Открытые файловые дескрипторы (fd)

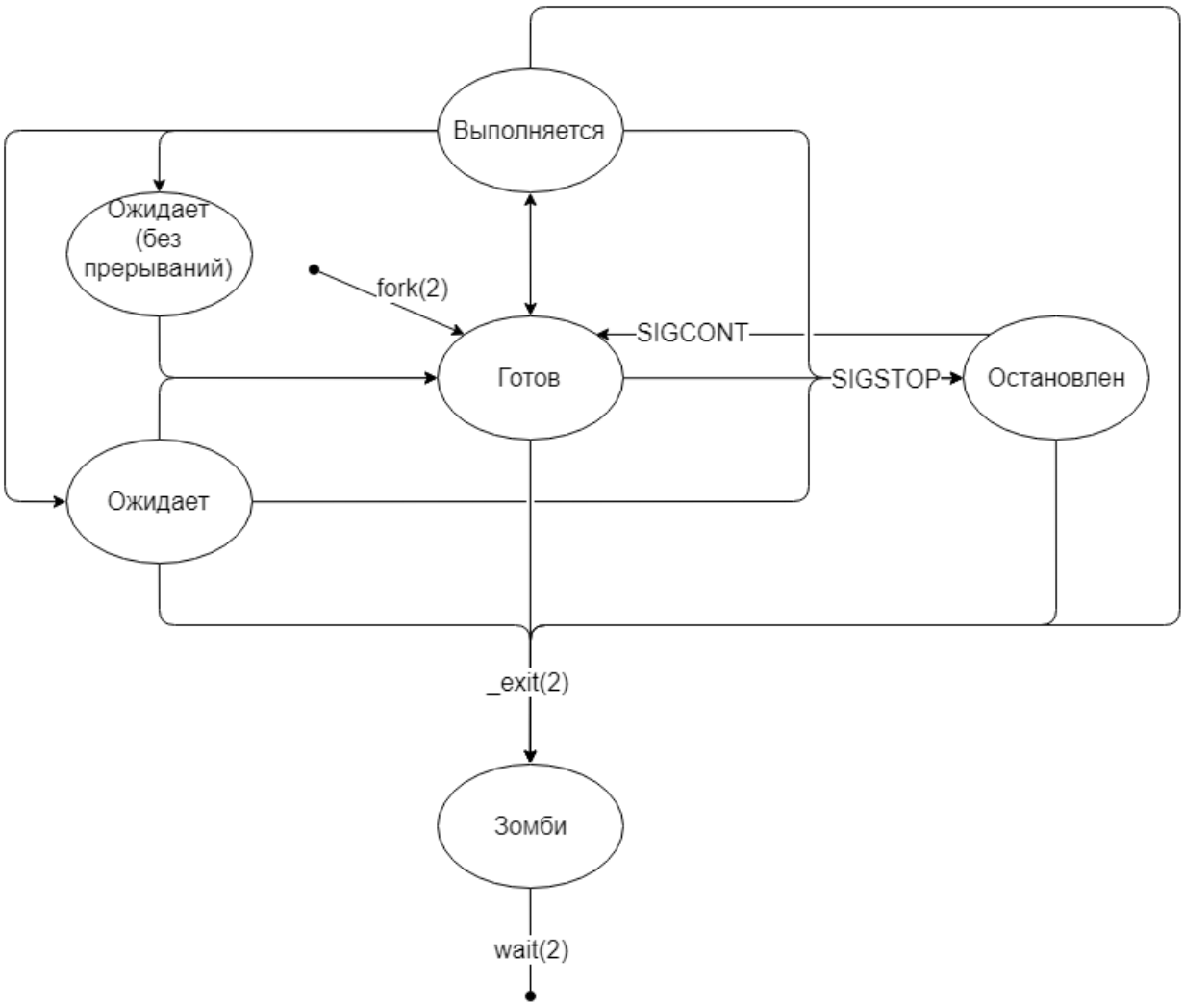
Обработчики сигналов (signal handler)

Текущий рабочий каталог (cwd)

Переменные окружения (environ)

Код возврата

Жизненный цикл процесса



Рождение процесса

Только один процесс в системе рождается особенным способом — init — он порождается непосредственно ядром. Все остальные процессы появляются путём дублирования текущего процесса с помощью системного вызова fork(2). После выполнения fork(2) получаем два практически идентичных процесса за исключением следующих пунктов:

fork(2) возвращает родителю PID ребёнка, ребёнку возвращается 0;

У ребёнка меняется PPID (Parent Process Id) на PID родителя.

После выполнения fork(2) все ресурсы дочернего процесса — это копия ресурсов родителя. Копировать процесс со всеми выделенными страницами памяти — дело дорогое, поэтому в ядре Linux используется технология Copy-On-Write.

Все страницы памяти родителя помечаются как read-only и становятся доступны и родителю, и ребёнку. Как только один из процессов изменяет данные на определённой странице, эта страница не изменяется, а копируется и изменяется уже копия. Оригинал при этом «отвязывается» от данного процесса. Как только read-only оригинал остаётся «привязанным» к одному процессу, странице вновь назначается статус read-write.

Ссылки на статьи

Хабр:

https://habr.com/ru/articles/423049/

Oppennet:

https://www.opennet.ru/docs/RUS/lnx_process/process2.html

Easy network:

https://easy-network.ru/uroki-linux/104-urok-19-protsessy-v-linux-komandy-prosmotra-i-upravleniya.html

Какой-то народ:

https://dmilvdv.narod.ru/Translate/ELSDD/elsdd_process_creation.html#:~:text=%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5%20%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%B0%20%D0%B2%20Linux%20%D0%BE%D1%81%D1%83%D1%89%D0%B5%D1%81%D1%82%D0%B2%D0%BB%D1%8F%D0%B5%D1%82%D1%81%D1%8F,%2C%20%D0%B8%D0%BC%D0%B5%D1%8E%D1%89%D0%B8%D0%BC%D0%B8%20%D1%80%D0%B0%D0%B7%D0%BD%D1%8B%D0%B5%20PID%2D%D1%8B.