

## Работа с процессами в операционной системе Linux

В операционной системе *Linux* для каждой отдельной программы, при ее запуске создается процесс. По сути, процесс является некоторым отображением каждого исполняемого программного кода. Однако, следует отметить, что в современных операционных системах каждая программа может одновременно работать с несколькими процессами и наоборот один процесс может использоваться несколькими программами.

В рамках процесса программе (программному коду) выделяется процессорное время, оперативная память и другие системные ресурсы. Также каждый процесс – с точки зрения работы процессора содержит свои наборы регистров и свой стек. Такая работа с процессами возможна так как происходит в виртуальном пространстве оперативной памяти, включающем как собственно оперативную память, так файл подкачки, разбитые при помощи и так называемый страничной модели.

В операционной системе *Linux* каждый процесс характеризуется своим уникальным идентификатором (*PID = Process ID*,). Первая запускаемая программа - программа инициализации получает *PID 1*, а каждая следующая запущенная программа - на единицу больше. Таким образом *PID* пользовательских программ доходит до нескольких тысяч. При этом следует отметить, что каждый процесс рождается как разветвление (*fork()*) предыдущего процесса (который называется родительским). При этом созданный процесс (дочерний) всегда содержит ссылку на *PID* родительского процесса (*PPID – parent PID*).

Каждый процесс помимо *PID* и *PPID* характеризуется следующими параметрами:

- *UID* - пользователь, от имени которого запущен процесс;
- *C* - процент времени *CPU*, используемого процессом;
- *STIME* - время запуска процесса;
- *TTY* - терминал, из которого запущен процесс;
- *TIME* - общее время процессора, затраченное на выполнение процессора;
- *CMD* - команда запуска процессора;
- *LWP* - показывает потоки процессора;
- *USER* - пользователь, от которого был запущен процесс;
- *NI* - приоритет выполнения процесса от -20 до 19;
- *PRI* - приоритет процесса *linux* на уровне ядра (обычно *NI+20*) ;
- *STAT* - Текущий статус процесса: *R* — выполняется; *D* — ожидает записи на диск; *S* — неактивен (< 20 с); *T*— приостановлен; *Z* — зомби;
- *CPU* - используемые ресурсы процессора;
- *MEM* - использованная память;

- **VSZ** - Виртуальный размер процесса;
- **RSS** - Размер резидентного набора (количество страниц памяти).

Каждый процесс может находиться в одном из нескольких состояний:

- **запуск** - процесс работает, или готов к работе и ожидает, когда ему будет дано процессорное время;
- **ожидание** - процессы в этом состоянии ожидают какого-либо события или освобождения системного ресурса. Ядро делит такие процессы на два типа - те, которые ожидают освобождения аппаратных средств и приостановление с помощью сигнала;
- **остановлено** - обычно, в этом состоянии находятся процессы, которые были остановлены с помощью сигнала;
- **зомби** - процессы были остановлены и больше не выполняются, но для них есть запись в таблице процессов, так как возможно остались дочерние процессы.

Также все процессы делимы на обычные и фоновые.

Файлы, соответствующие процессам, находятся в каталоге */proc*.

Например, первый процесс – процесс инициализации хранится в каталоге */proc/1/*.

Более простым и удобным способом посмотреть процессы в *linux* является команда *ps*.

Следует отметить, что запрос *ps* поддерживает два типа ключей *unix* и ключи *BSD*. Далее будут использованы ключи *unix*.

Запрос *ps* без ключей выводит только процессы текущей оболочки. Все процессы, запущенные в операционной системе, могут быть просмотрены при помощи запроса

*ps -A* (также возможен запрос *ps -e*)

также можно вывести все процессы в виде дерева:

*ps -AH*

Более удобное отображение дерева процессов может быть получено при помощи утилиты *pstree*.

Чтобы вывести больше информации о ресурсах процессов используйте опцию *-f*:

*ps -f* или *ps -F* (информация, включая занимаемую память).

Так, например можно вывести подробную информацию обо всех процессах:

*ps -Af*

можно отобразить все процессы, запущенные от имени определенного пользователя:

*ps -fu root*

если необходима информация только о процессе по его *pid*, используйте флаг *-p*:

*ps -fp 1*

Опция *-C* позволяет фильтровать процессы по имени, например, выберем только процессы *firefox*:

*ps -fC firefox*

также поиск процесса можно выполнить, например, командой *grep*, например запроса

*ps -f | grep firefox*

Кроме основных флагов утилиты *ps* допускает также и дополнительные флаги, так, например, запрос

*ps -Af --sort=%mem*

позволяет вывести все процессы, сортированные по % памяти.

Следующий запрос позволяет вывести процессы, сортированные по проценту загрузки *cpu*:

*ps -FA --sort pcpu*

### **Другие ключи**

- A - выбрать все процессы;
- a - выбрать все процессы, кроме фоновых;
- d, (g) - выбрать все процессы, даже фоновые, кроме процессов сессий;
- N - выбрать все процессы кроме указанных;
- C - выбирать процессы по имени команды;
- G - выбрать процессы по *ID* группы;
- p, (p) - выбрать процессы *PID*;
- ppid - выбрать процессы по *PID* родительского процесса;
- s - выбрать процессы по *ID* сессии;
- t, (t) - выбрать процессы по *tty*;
- u, (U) - выбрать процессы пользователя.
- Z (M) – информация о безопасности процессов.

### **Опции форматирования:**

- c - отображать информацию планировщика;
- f - вывести максимум доступных данных, например, количество потоков;
- j, (j) - вывести процессы в стиле *Jobs*, минимум информации;
- M, (Z) - добавить информацию о безопасности;
- o, (o) - позволяет определить свой формат вывода;
- sort, (k) - выполнять сортировку по указанной колонке;
- L, (H) - отображать потоки процессов в колонках *LWP* и *NLWP*;

- m*, (*m*) - вывести потоки после процесса;
- V*, (*V*) - вывести информацию о версии;
- H* - отображать дерево процессов *linux*;

Чтобы узнать идентификатор того или иного процесса можно воспользоваться командой *pgrep*, например:

```
pgrep bash.
```

### **Настройка процессов**

**Отслеживание процессов в реальном времени.** Для отслеживания процессов используется команда *top*. Которая позволяет отображать в терминале текущее состояние процессов и все его изменения в отличии от *ps* – где состояние системы запоминается на момент запроса.

В верхней части результата работы *top* предоставлена статистика работы системы, (нагрузка на систему и число выполняемых задач). В нижней части отображается информация по запущенным процессам и статистики их использования.

Кроме того, в интерактивном режиме утилита *top* принимает команды для управления выводом: команды завершения процесса по его *PID*: *k*, команды сортировки по пользователю *U*.

**Приоритет процесса.** Приоритет процесса *linux* означает, насколько больше процессорного времени будет отдано этому процессу по сравнению с другими. Так можно очень тонко настроить какая программа будет работать быстрее, а какая медленнее. Значение приоритета может колебаться от 19 (минимальный приоритет) до -20 - максимальный приоритет процесса *linux*. Уменьшать приоритет можно с правами обычного пользователя, но, чтобы его увеличить нужны права суперпользователя. Для регулировки приоритета используется команда *nice*. С помощью нее вы можете указать приоритет для запускаемого процесса:

```
nice -n 10 yum
```

или, например, изменить приоритет для существующего процесса по его *pid*:

```
renice -n 10 -p 1343
```

**Ресурсы процессов.** Узнать о текущих ограничениях по ресурсам и поменять их для текущей оболочки можно при помощи команды *ulimit*. Запрос

```
ulimit -a pid
```

позволяет узнать о текущих ограничениях. Однако сами ограничения для каждого процесса хранятся в файлах

```
cat /proc/< pid >/limits.
```

**Завершение и сигналы процессов.** Каждый процесс может быть принудительно завершен при помощи утилиты *kill*. При выполнении данной команды всем процессам отправляется *TERM*-сигнал. Таким образом, рабочая программа выполняет требуемые операции по очистке и безопасно завершает работу. Запрос по удалению процесса может быть выполнен следующим образом:

*kill -TERM 1943*

Также можно уничтожить процесс по имени:

*killall firefox*

Следует отметить, что на самом деле утилита *kill* поддерживает широкий перечень сигналов, список которых может быть получен по следующему запросу :

*Kill -l*

Так, например, выше описанный сигнал *TERM* (*SIGTERM*) – является сигналом мягкого завершения работы процесса. Однако, в случае необходимости может быть использован сигнал *KILL* (*SIGKILL*), обеспечивающий “грубое” завершение процесса. Все порожденные процессы останутся незавершенными - это может привести к непредсказуемым последствиям. Поэтому данный способ лучше применять лишь в крайних случаях.

Следует отметить, что помимо наименования сигнала может быть использовано его цифровое обозначение, так сигналы

*kill -9 PID* и *kill KILL PID* будут идентичными.

Таблица 1 – список некоторых из наиболее популярных сигналов процессов

SIGNAL NAME	SIGNAL NUMBER	DESCRIPTION
SIGHUP	1	Hang up detected on controlling terminal or death of controlling process
SIGINT	2	Issued if the user sends an interrupt signal (Ctrl + C).
SIGQUIT	3	Issued if the user sends a quit signal (Ctrl + D).
SIGFPE	8	Issued if an illegal mathematical operation is attempted
SIGKILL	9	If a process gets this signal it must quit immediately and will not perform any clean-up operations
SIGALRM	14	Alarm Clock signal (used for timers)
SIGTERM	15	Software termination signal (sent by kill by default).

### Задание 1.

1. откройте браузер.
2. найдите *PID* соответствующего процесса.
3. поменяйте приоритет процесса браузера, верните его.
4. откройте утилиту *top*, найдите в ней браузер.
5. закройте браузер через утилиту *top*.
6. выведите все процессы, сортированные по объему памяти,
7. выведите все процессы по объему ресурсов процессора в виде дерева.

### Задание 2.

1. Откройте два окна терминала (*левый* и *правый*). В левом окне запустите бесконечный процесс, который с интервалом в одну секунду будет писать строку AAA в файл *test.out* в домашнем каталоге.

```
$ ( while true; do printf "AAA %d " $$ >> ~/test.out; sleep 1; done )
```

2. В правом окне наблюдайте за работой процесса.

```
$ tail -f ~/test.out
```

3. В левом окне остановите процесс (*Ctrl+z*). *BASH* вернет номер задания в квадратных скобках. Убедитесь, что процесс перестал писать в файл – какое у процесса состояние?

4. В левом окне выведите список заданий (*jobs*). Определите текущее задание. Определите состояние процессов.

5. Определите функционал следующих команд и поэкспериментируйте с ними:

```
$ bg; $ fg.
```

6. Определите функционал следующих команд:

```
$ jobs; $ ps j.
```

7. Запустите задание в фоновом режиме (знак «&» после объявления). Убедитесь, что процесс снова продолжил писать в файл.

8. Определите состояние процесса во время его работы.

9. Заверите процесс командой *kill -TERM* – что при этом изменится в заданиях процессов.

10. Запустите задание в фоновом режиме и заверите процесс командой *kill -KILL* – что при этом изменится в заданиях процессов.

### Задание 3.

1. Запустите два бесконечных процесса, которые с интервалом в одну секунду будут писать строки в файл *test.out* в домашнем каталоге.
2. Выведите список заданий (наблюдайте *Running*). Определите состояние процессов (*jobs* и *ps j*).
3. Пронаблюдайте за работой процессов.
4. Остановите первый процесс (задание "AAA") при помощи запроса  
*kill -SIGSTOP %1*.  
Выведите список заданий (наблюдайте *Stopped*).
5. Завершите второй процесс (задание "UUU"):  
*kill -SIGTERM %2*
6. Выведите список заданий (наблюдайте *Terminated*).
7. В левом окне завершите процесс.  
*\$ fg*  
*\$ Ctrl+c*
8. В правом окне завершите команду *tail*.

### Задание 4.

1. Создайте файл *test-trap.sh*
2. При помощи редактора *vi* запишите в файл следующий сценарий:  

```
$ cat test-trap.sh
#!/bin/bash
declare -i i=0
trap 'echo "Аварийное завершение..."; exit' SIGINT
while [ $i -lt 100 ]
do
    (( i++ ))
    echo $i
    sleep 1
done
```
3. Сделайте файл исполняемым  
*chmod +x test-trap.sh*
4. Проверьте процессы в работе и запустите файл  
*./test-trap.sh*
5. проверьте процессы и объясните, что поменялось и почему.
6. Прервите файл при помощи комбинации клавиш *ctrl+C*, объясните что произошло.

### Задание 5.

1. Изучите содержимое файла */proc/version* и сравните с выводом команды *uname -a*.
2. Изучите и сравните содержимое файлов  
*/proc/meminfo* и */sys/devices/system/node/node0/meminfo*,  
и вывод команды *free*.
3. Изучите содержимое файла  
*/proc/cpuinfo*.  
Определите количество ядер.
4. Изучите содержимое файла  
*/proc/uptime*  
и сравните с выводом команды *uptime*.
5. Изучите специальную символьную ссылку */proc/self*, которая указывает на подкаталог текущего процесса:  
*\$ echo \$\$*  
Объясните результаты.  
Изучите ресурсы оболочки при помощи запроса  
*ulimit -a*
6. Изучите ресурсы процесса инициализации при помощи запроса  
*ulimit -a 1*  
сравните результат с данными файла */proc/1/limits*