***https://hostovita.pl/blog/zarzadzanie-procesami-linux-ps-kill-nice/***

**Jak zobaczyć działające procesy w Linux**

**top**

Najprostszym sposobem, aby dowiedzieć się, jakie procesy są uruchomione na naszym serwerze – jest wykonanie polecenia top:  
top

top - 15:14:40 up 46 min, 1 user, load average: 0.00, 0.01, 0.05

Tasks: 56 total, 1 running, 55 sleeping, 0 stopped, 0 zombie

Cpu(s): 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st

Mem: 1019600k total, 316576k used, 703024k free, 7652k buffers

Swap: 0k total, 0k used, 0k free, 258976k cached

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

1 root 20 0 24188 2120 1300 S 0.0 0.2 0:00.56 init

2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd

3 root 20 0 0 0 0 S 0.0 0.0 0:00.07 ksoftirqd/0

6 root RT 0 0 0 0 S 0.0 0.0 0:00.00 migration/0

7 root RT 0 0 0 0 S 0.0 0.0 0:00.03 watchdog/0

8 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 cpuset

9 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 khelper

10 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kdevtmpfs

Górna część odpowiedzi pokazuje nam statystyki systemu, takich jak obciążenie systemu i ilość aktywnych zadań

Widzimy, że mamy 1 aktywny proces i 55 śpiących (które nie wykorzystują zasobów procesora).

Poniżej znajduje się lista uruchomionych procesów ze statystykami użytkowania.

**htop**

Ulepszona wersją top nazywa się htop. Jest ona dostępna w repozytorium.  
Instalujemy ją:  
sudo apt-get install htop  
  
Jak tylko htop zostanie zainstalowany możemy sprawdzić odpowiedź tego narzędzi:  
htop

Mem[||||||||||| 49/995MB] Load average: 0.00 0.03 0.05

CPU[ 0.0%] Tasks: 21, 3 thr; 1 running

Swp[ 0/0MB] Uptime: 00:58:11

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command

1259 root 20 0 25660 1880 1368 R 0.0 0.2 0:00.06 htop

1 root 20 0 24188 2120 1300 S 0.0 0.2 0:00.56 /sbin/init

311 root 20 0 17224 636 440 S 0.0 0.1 0:00.07 upstart-udev-brid

314 root 20 0 21592 1280 760 S 0.0 0.1 0:00.06 /sbin/udevd --dae

389 messagebu 20 0 23808 688 444 S 0.0 0.1 0:00.01 dbus-daemon --sys

407 syslog 20 0 243M 1404 1080 S 0.0 0.1 0:00.02 rsyslogd -c5

408 syslog 20 0 243M 1404 1080 S 0.0 0.1 0:00.00 rsyslogd -c5

409 syslog 20 0 243M 1404 1080 S 0.0 0.1 0:00.00 rsyslogd -c5

406 syslog 20 0 243M 1404 1080 S 0.0 0.1 0:00.04 rsyslogd -c5

553 root 20 0 15180 400 204 S 0.0 0.0 0:00.01 upstart-socket-br

**Jak wyświetlić listę procesów przy pomocy ps**

top i htop posiadają przyjazny interfejs do podglądu działających procesów, który jest podobny do graficznych menedżerów zadań.

Jednak te narzędzia nie zawsze są na tyle elastyczne, aby potrafić dokonać wszelkich działań. Ale na szczęście mamy narzędzie ps, które może dopełnić funkcjonalność poprzednich narzędzi.

Bez podania argumentów, odpowiedź będzie podobna do takiej:  
ps

PID TTY TIME CMD

1017 pts/0 00:00:00 bash

1262 pts/0 00:00:00 ps

W odpowiedzi będą znajdować się wszystkie procesy, związane z bieżącym użytkownikiem i aktywną sesją naszego terminalu. Taka odpowiedź ma sens, bo w tej chwili mamy działające tylko bash i ps.

Aby uzyskać więcej informacji odnośnie uruchomionych procesów na komputerze czy serwerze, można wykonać następne polecenie:  
ps aux

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

root 1 0.0 0.2 24188 2120 ? Ss 14:28 0:00 /sbin/init

root 2 0.0 0.0 0 0 ? S 14:28 0:00 [kthreadd]

root 3 0.0 0.0 0 0 ? S 14:28 0:00 [ksoftirqd/0]

root 6 0.0 0.0 0 0 ? S 14:28 0:00 [migration/0]

root 7 0.0 0.0 0 0 ? S 14:28 0:00 [watchdog/0]

root 8 0.0 0.0 0 0 ? S< 14:28 0:00 [cpuset]

root 9 0.0 0.0 0 0 ? S< 14:28 0:00 [khelper]

. . .

Ta opcja mówi ps, aby pokazać procesy wszystkich użytkowników, niezależnie od terminalu w przyjaznym dla użytkownika formacie.

Aby zobaczyć całe drzewo, hierarchie procesów, możemy skorzystać z takich opcji:  
ps axjf

PPID PID PGID SID TTY TPGID STAT UID TIME COMMAND

0 2 0 0 ? -1 S 0 0:00 [kthreadd]

2 3 0 0 ? -1 S 0 0:00 \\_ [ksoftirqd/0]

2 6 0 0 ? -1 S 0 0:00 \\_ [migration/0]

2 7 0 0 ? -1 S 0 0:00 \\_ [watchdog/0]

2 8 0 0 ? -1 S< 0 0:00 \\_ [cpuset]

2 9 0 0 ? -1 S< 0 0:00 \\_ [khelper]

2 10 0 0 ? -1 S 0 0:00 \\_ [kdevtmpfs]

2 11 0 0 ? -1 S< 0 0:00 \\_ [netns]

. . .

Jak widać z odpowiedzi, proces kthreadd jest procesem nadrzędnym procesu ksoftirqd/0 i innych.

**Identyfikatory procesów**

W systemach Linux i Unix`owych dla każdego procesy jest przypisywany unikatowy identyfikator **process ID** czy **PID**. Tak system operacyjny identyfikuje i śledzi za procesami.

Najszybszym sposobem uzyskać identyfikator procesy jest skorzystanie z pgrep:  
pgrep bash

1017

To było zwykłe zapytanie o identyfikator procesu.

Pierwszym procesem, który uruchamia się podczas ładowania systemu, nazywa się init, i on posiada PID „1”.  
pgrep init

1

Ten proces odpowiada za włączenie wszystkich innych w systemie. Kolejne procesy będą mieli wyższe numery PID.

Nadrzędny proces jest procesem, który odpowiada za swoich „dzieci”. Jak tylko zabijemy proces nadrzędny – wszytkie podrzędne tego procesu też przestaną działać. Nadrzędny proces zamiast PID – posiada swój **PPID**.

Możesz zobaczyć PID i PPID w nagłówkach prawie wszystkich narzędzi do zarządzania procesami, w tym top, htop i ps.

Komunikacja w sprawie procesów pomiędzy użytkownikiem a systemem odbywa się przez tłumaczenie nazwy procesu oraz jego PID. Dlatego programy podpowiadają nam PID.

**Jak wysyłać sygnały do procesów w systemach Linux**

Wszystkie procesy w Linux reagują na sygnały. Takie sygnały, wysyłane do procesów, to jeden ze sposobów powiedzieć procesowi na poziomie systemu operacyjnego przerwać się lub zmienić swoje zachowanie.

**Jak wysyłać sygnały do procesów przez PID**

Najbardziej popularnym sposobem przesyłania sygnałów do programów – jest kill.

Jak można się domyślić, domyślna funkcjonalność tej funkcji może zakończyć pracę procesu:  
kill PID\_procesu  
Takie polecenie wysyła sygnał **TERM** do wykonywanego procesu. TERM prosi u procesu grzecznie zakończyć pracę. To pozwoli dokonać czyszczenie działających procesów i wyjść bez dodatkowych problemów.

Jeśli program zachowuje się nie tak, jak oczekiwaliśmy, i nie chce zakończyć swoją pracę, możemy wymusić to przez parametr KILL:  
kill -KILL PID\_procesu  
Jest to specjalny sygnał, który wysyła się nie do programu czy procesu.

Zamiast tego, on zwraca się do najniższej warstwy systemu, który ze swojej strony wstrzymuje ten proces. Wykorzystuje się to w celu ominięcia blokad, które wykorzystują programy.

Każdy sygnał posiada odpowiedni numer, który można przekazać zamiast jego nazwy. Na przykład, można skorzystać z „-15” zamiast „-TERM”, i „-9” zamiast „-KILL”.

**Jak skorzystać z sygnałów do innych cełów**

Sygnały są używane nie tylko do wyłączania programów i procesów. Mogą one również wykonywać inne działania.

Na przykład, wiele demonów może się zrestartować kiedy podamy sygnał HUP. Apache jest jednym z takich programów:  
sudo kill -HUP pid\_apache

Takie polecenie spowoduje ponowne uruchomienie i wczytywanie pliku konfiguracyjnego Apache.

Można otrzymać listę wszystkich sygnałów, które można wysłać z poleceniem kill:  
kill -l

1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL 5) SIGTRAP

6) SIGABRT 7) SIGBUS 8) SIGFPE 9) SIGKILL 10) SIGUSR1

11) SIGSEGV 12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM

. . .

**Jak wysłać sygnał z wykorzystaniem nazwy procesu**

Chociaż tradycyjnym sposobem wysyłania sygnałów jest wykorzystanie numerów PID procesów, istnieją również sposoby z wykorzystaniem standardowych nazw procesów.

W pkill polecenia działają praktycznie tak samo, jak i w kill, ale działa to na nazwach procesów, a nie ich identyfikatorach:  
pkill -9 ping  
To polecenie wykonuje to samo, co i poniższe:  
kill -9 `pgrep ping`  
Jeżeli chcesz wysłać sygnał na każdy podrzędny proces, można użyć killall:  
killall firefox  
Powyższe polecenie wyszli sygnały TERM do każdej działającej instancji firefox na serwerze czy komputerze.

**Jak zmienić priorytet procesu**

Czasami chcemy, aby niektóre procesy mieli większy priorytet w środowisku serwera.

Niektóre z nich mogą być krytycznie ważnymi w pewnych sytuacjach, a inne mogą być wykonywane w przypadku dostępnych zasobów serwera.

Linux zarządza priorytetami procesów przez wartość **niceness**.

Zadania o wysokim priorytecie są uznawane za mniej przyjemne (nice), bo nie mają zasobów. Zadania z mniejszym priorytetem są bardziej przyjazne, bo oni wymagają mniej zasobów i mocy serwera.

Kiedy na początku sprawdziliśmy system za pomocą top, zobaczyliśmy kolumnę „NI”. Co oznacza, że proces ma dobrą wartość:  
top

Tasks: 56 total, 1 running, 55 sleeping, 0 stopped, 0 zombie

Cpu(s): 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st

Mem: 1019600k total, 324496k used, 695104k free, 8512k buffers

Swap: 0k total, 0k used, 0k free, 264812k cached

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

1635 root 20 0 17300 1200 920 R 0.3 0.1 0:00.01 top

1 root 20 0 24188 2120 1300 S 0.0 0.2 0:00.56 init

2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd

3 root 20 0 0 0 0 S 0.0 0.0 0:00.11 ksoftirqd/0

Dobre lub przyjazne wartości mogą być w przedziale między „-19/-20” (najwyższy priorytet) i „19/20” (najniższy priorytet) w zależności od systemu.

Aby uruchomić program z dobrze określonym priorytetem, możemy użyć polecenie nice:  
nice -n 15 polecenie\_do\_wykonania  
Takie polecenie działa tylko dla uruchomienia programu.

Żeby zmienić wartość priorytetu *nice* dla już działającego programu czy procesu – musimy skorzystać z narzędzia renice:  
renice 0 PID\_dla\_zwiększenia\_priorytetu  
**Uwaga**: *nice* działa z nazwami programów, a *renice* już z identyfikatorami procesów **PID**.