Monitorowanie na żywo: top, htop, btop

Poziom: podstawy **Dotyczy: RHEL/Rocky 9–10, różnice Debian/Ubuntu i SLES zaznaczone**

Czas: ~16 min

`top` i `htop` robią to, czego `ps` nie robi: pokazują system na żywo, odświeżając widok co kilka sekund. Ta strona uczy czytać nagłówek `top` (load average, `wa`, `st`) i naciskać właściwe klawisze, zanim zabijesz nie ten proces.

Skróty użyte na tej stronie (8)

Skrót	Co znaczy
CPU	Central Processing Unit: procesor
EPEL	Extra Packages for Enterprise Linux: dodatkowe repozytorium pakietów dla RHEL i klonów
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
PID	Process ID: numer procesu w systemie
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SLES	SUSE Linux Enterprise Server: płatna dystrybucja Linuksa firmy SUSE, konkurent Rocky/RHEL
VM	Virtual Machine: maszyna wirtualna
WMS	Warehouse Management System: system zarządzania magazynem

Fiszka

TYLKO ODCZYT

Uruchomienie interaktywne

```
top
```

Domyślnie sortujemy malejąco po zużyciu CPU, a widok odświeża się co 3 sekundy, dopóki nie naciśniemy `q`. Górne pięć linii to nagłówek systemowy, poniżej lista procesów.

Przykład:

```
top - 14:32:07 up 23 days,  4:12,  2 users,  load average: 3.41, 2.87, 1.02
Tasks: 187 total,   2 running, 184 sleeping,   0 stopped,   1 zombie
%Cpu(s): 22.1 us,  4.3 sy,  0.0 ni, 58.2 id, 14.8 wa,  0.0 hi,  0.6 si,  0.0 st
MiB Mem :  15872.0 total,   1203.4 free,   9821.7 used,   4846.9 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   5432.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1203 postgres  20   0 2481200 943212  81204 S   1.3   5.9   3:12.44 postgres
   4821 nginx     20   0   55600   8420   6210 S   0.3   0.1   0:03.21 nginx
```

CPU = Central Processing Unit: procesor · **top** = table of processes: procesy na żywo

TYLKO ODCZYT

Filtrowanie procesów jednego użytkownika: `-u`

```
top -u nazwa_usera
```

Zawężamy listę procesów do jednego użytkownika, nagłówek systemowy zostaje bez zmian — obciążenie i pamięć dotyczą zawsze całej maszyny, nie da się ich filtrować.

top = table of processes: procesy na żywo

TYLKO ODCZYT

Monitorowanie konkretnych PID-ów: `-p`

```
top -p 1234,5678
```

Śledzimy tylko wskazane procesy zamiast całej listy — wygodne, gdy podejrzewamy konkretną usługę i nie chcemy, żeby ekran skakał przy każdym odświeżeniu. Pułapka opisana w *Częste błędy*: lista PID-ów jest zamrożona na starcie.

top = table of processes: procesy na żywo

TYLKO ODCZYT

Wykonanie pojedynczego zrzutu zamiast interfejsu: `-b -n 1`

```
top -b -n 1 | head -8
```

`-b` (batch) wyłącza interfejs interaktywny i drukuje zwykły tekst, `-n 1` każe zrobić dokładnie jedno odświeżenie i zakończyć. Wersja `top` do crona i skryptów diagnostycznych — bez `-n` `top -b` leciałby w nieskończoność, zapychając log.

Przykład:

```
top - 14:40:02 up 23 days,  4:20,  2 users,  load average: 2.10, 2.55, 1.80
Tasks: 187 total,   1 running, 185 sleeping,   0 stopped,   1 zombie
%Cpu(s): 12.4 us,  3.1 sy,  0.0 ni, 83.0 id,  1.4 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :  15872.0 total,    980.2 free,  10120.4 used,   4771.4 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   5210.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1203 postgres  20   0 2481200 943212  81204 S   1.1   5.9   3:12.55 postgres
```

top = table of processes: procesy na żywo

TYLKO ODCZYT

Sortowanie od startu według pamięci: `-o %MEM`

```
top -b -n 1 -o %MEM | head -8
```

Bez tej flagi trzeba by nacisnąć `M` już po uruchomieniu; `-o` ustawia kolumnę sortującą od razu, co ma znaczenie zwłaszcza w trybie wsadowym, gdzie nie ma okazji nacisnąć czegokolwiek. Widać tu, że teraz `postgres` (najwyższy `%MEM`) prowadzi, nawet jeśli akurat nie zużywa najwięcej CPU.

Przykład:

```
top - 14:41:15 up 23 days,  4:21,  2 users,  load average: 2.05, 2.50, 1.79
Tasks: 187 total,   1 running, 185 sleeping,   0 stopped,   1 zombie
%Cpu(s): 11.0 us,  2.8 sy,  0.0 ni, 84.8 id,  1.3 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :  15872.0 total,    975.1 free,  10125.0 used,   4771.9 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   5205.4 avail Mem

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

```
 1203 postgres  20   0 2481200 943212  81204 S   0.9   5.9   3:12.61 postgres
 4821 nginx     20   0   55600   8420   6210 S   0.0   0.1   0:03.21 nginx
```

CPU = Central Processing Unit: procesor · **top** = table of processes: procesy na żywo

TYLKO ODCZYT

Skróty klawiszowe w `top` i `htop`

W obu narzędziach naciskasz klawisz w trakcie działania, bez wychodzenia z ekranu.

Klawisze w `top`:

Klawisz	Robi
`1`	rozbija zbiorczy wiersz CPU na osobny wiersz per rdzeń
`M`	sortuj wg zużycia pamięci
`P`	sortuj wg zużycia CPU (domyślne)
`T`	sortuj wg czasu CPU (TIME+)
`k`	zabij proces — pyta o PID, potem o sygnał
`r`	zmień priorytet (renice) działającego procesu
`u`	filtruj po użytkowniku
`c`	pokaż pełną ścieżkę/linię poleceń zamiast samej nazwy
`H`	pokaż wątki jako osobne wiersze (jak `ps -eLf`)
`V`	widok drzewa procesów (forest)
`W`	zapisz bieżącą konfigurację do `~/.config/procps/toprc`
`d`	zmień częstotliwość odświeżania
`q`	wyjście

Klawisze w `htop`:

Klawisz	Robi
`F1` / `h`	pomoc
`F2`	ustawienia (kolory, kolumny, paski)
`F3`	wyszukaj proces po nazwie
`F4`	filtruj listę (wpisujesz fragment nazwy)
`F5`	widok drzewa procesów

Klawisz	Robi
`F6`	wybór kolumny sortowania
`F9`	wyślij sygnał do zaznaczonego procesu
`Spacja`	zaznacz proces (można wiele naraz, potem `F9` na wszystkie)
`u`	filtruj po użytkowniku

CPU = Central Processing Unit: procesor · **PID** = Process ID: numer procesu w systemie

TYLKO ODCZYT

`htop`

```
htop
```

Interaktywny odpowiednik `top` z kolorami, paskami zużycia CPU/RAM per rdzeń i obsługą myszki. Na RHEL/Rocky wymaga włączenia EPEL — instalacja opisana w *Krok po kroku*.

CPU = Central Processing Unit: procesor · **RHEL** = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **EPEL** = Extra Packages for Enterprise Linux: dodatkowe repozytorium pakietów dla RHEL i klonów

TYLKO ODCZYT

Filtrowanie procesów użytkownika w `htop`: `-u`

```
htop -u nazwa_usera
```

Ten sam filtr co w `top -u`, ale w interfejsie, w którym dodatkowo zawężamy listę dynamicznie klawiszem `F4`.

TYLKO ODCZYT

`btop`

```
btop
```

Nowsza alternatywa dla `top`/`htop` — wykresy zamiast samych liczb, osobne panele CPU/pamięć/sieć/dysk. Nie jest częścią żadnej dystrybucji z tej strony w domyślnych repozytoriach; instaluje się z EPEL, z repozytorium producenta albo jako pakiet statyczny.

□□ **CPU** = Central Processing Unit: procesor · **EPEL** = Extra Packages for Enterprise Linux: dodatkowe repozytorium pakietów dla RHEL i klonów

□□ TYLKO ODCZYT

Wykonanie cyklicznego raportu zamiast ekranu na żywo: `pidstat`

```
pidstat 1 5
```

Część pakietu `sysstat`. Zamiast odświeżanego widoku drukujemy kolejne próbki tekstowe — co 1 sekundę, 5 razy — co nadaje się do logowania i dalszego przetwarzania, w odróżnieniu od `top` / `htop`.

Przykład:

```
Linux 5.14.0-427.el9.x86_64 (srv01)     09/17/2026      _x86_64_    (4 CPU)

14:32:10      UID       PID    %usr %system  %guest   %wait    %CPU   CPU  Command
14:32:11      989      4821    0.99    0.00    0.00    0.00    0.99     1  nginx
14:32:11        0      1203    1.98    0.99    0.00    0.00    2.97     2  postgres
```

□□ TYLKO ODCZYT

Odczytanie PID po nazwie, bez interfejsu: `pidof`

```
pidof nginx
```

Najprostszy sposób na uzyskanie samych numerów PID w skrypcie, bez parsowania wyniku `ps` czy `top`.

Przykład:

```
4823 4822 4821
```

□□ **PID** = Process ID: numer procesu w systemie

□□ TYLKO ODCZYT

Uruchomienie z innym priorytetem: `nice`

```
nice -n 10 ./skrypt.sh
```

Uruchamiamy proces z podwyższoną wartością `nice`, czyli niższym priorytetem planisty — proces ustępuje miejsca innym, gdy o CPU robi się ciasno. Samo `nice` nic nie wypisuje — wynik widać dopiero w `top` / `ps`.

CPU = Central Processing Unit: procesor

TYLKO ODCZYT

Zmiana priorytetu już działającego procesu: `renice`

```
renice -n 5 -p 1234
```

To samo co `nice`, ale dla procesu, który już działa — nie trzeba go restartować, żeby zmienić priorytet.

Przykład:

```
1234 (process ID) old priority 0, new priority 5
```

TYLKO ODCZYT

Ustawienie priorytetu w kolejce I/O: `ionice`

```
ionice -c 3 -p 1234
```

`nice` / `renice` regulują dostęp do CPU, `ionice` reguluje dostęp do dysku. Klasa `3` (idle) oddaje pierwszeństwo każdemu innemu procesowi — proces dostaje pasmo I/O tylko wtedy, gdy nikt inny go nie potrzebuje. Polecenie kończy się bez żadnego komunikatu.

CPU = Central Processing Unit: procesor

TYLKO ODCZYT

Wszystkie flagi

Flaga / komenda	Co robi
`top -u user`	tylko procesy jednego użytkownika

Flaga / komenda	Co robi
`top -p PID,...`	monitoruje wskazane PID-y (lista zamrożona na starcie)
`top -b -n N`	tryb wsadowy, dokładnie N odświeżeń
`top -o pole`	sortuje od startu wg wskazanego pola
`htop -u user`	jak wyżej, w htop
`pidstat N M`	M próbek co N sekund, CPU per proces
`pidof nazwa`	PID-y procesu po nazwie
`nice -n N komenda`	uruchamia z priorytetem CPU N
`renice -n N -p PID`	zmienia priorytet CPU działającego procesu
`ionice -c KLASA -p PID`	ustawia klasę priorytetu I/O

Uważaj

- `st` w wierszu `%Cpu` to czas skradziony przez hypervisor — wysoki `st` na VM oznacza, że problem jest poza Twoją maszyną, nie w niej.
- `top -p` z listą PID-ów zamrożoną raz nie dopisze nowych procesów, które pojawią się później — trzeba wyjść i wejść ponownie z nową listą.
- Na RHEL/Rocky `htop` **nie jest** w domyślnych repozytoriach — wymaga EPEL (szczegóły niżej).

PID = Process ID: numer procesu w systemie · **CPU** = Central Processing Unit: procesor · **VM** = Virtual Machine: maszyna wirtualna · **RHEL** = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **EPEL** = Extra Packages for Enterprise Linux: dodatkowe repozytorium pakietów dla RHEL i klonów

Z życia wzięte

Z ŻYCIA WZIĘTE

O 6:05, zaraz po starcie zmiany w Swarzędzu, Marek dzwoni, że WMS na ora01 ledwo odpowiada. W top widzimy load average powyżej ośmiu, ale kolumna wa pokazuje prawie zero, więc winny nie jest dysk, tylko sam procesor. Po chwili znajdujemy proces raportu uruchomiony przez pomyłkę o tej samej godzinie, który zjada cały CPU.

Po co to istnieje

`ps` daje zdjęcie. Zdjęcie nie mówi, czy proces, który w tej chwili zużywa 40% CPU, robi to od trzech sekund, czy od trzech godzin, ani czy zaraz zacznie zużywać 100%. Do tego potrzebny jest widok odświeżany w pętli — i dokładnie to robią `top` i jego następcy.

Drugi powód: `ps` pokazuje procesy, ale nie mówi nic o systemie jako całości — ile jest wolnego RAM-u, jak bardzo obciążony jest procesor, ile czasu system spędza czekając na dysk. Nagłówek `top` odpowiada na te pytania jednym rzutem oka, zanim jeszcze spojrzysz na listę procesów.

Analogia

`top` to **tablica przylotów na lotnisku**.

Odświeża się co kilka sekund, pokazuje stan teraz: który lot właśnie ląduje, który stoi na płycie, który jest opóźniony. Nie musisz stać przy tablicy cały dzień — wystarczy jedno spojrzenie, żeby wiedzieć, co się dzieje. Górna część tablicy — komunikaty ogólne, pogoda, alerty — odpowiada nagłówkowi `top` z load average i zużyciem CPU. Lista lotów poniżej odpowiada liście procesów.

I tu jest sedno analogii: **jedna kolumna na tej tablicy mówi więcej o kondycji całego lotniska niż cała reszta razem wzięta** — kolumna opóźnień. Jeśli połowa lotów ma status „opóźniony", nie obchodzi Cię już, który konkretnie samolot i dlaczego — wiesz, że problem jest systemowy (pogoda, awaria pasa, przeciążenie). W `top` tą kolumną jest `wa` (I/O wait) w wierszu `%Cpu`: jeśli jest wysoka, nieważne który proces akurat wygląda podejrzanie — problemem jest dysk albo sieć, nie konkretna aplikacja.

Jak to działa

Rozwiń: jak to działa pod spodem

Nagłówek `top`, linia po linii:

> **Pojęcie: obciążenie systemu (load average).** Load average to średnia liczba procesów, które w danej chwili są gotowe do działania na procesorze (stan running/runnable) albo czekają na zakończenie operacji I/O (stan D) — czyli praca, którą system *chciałby* wykonać, gdyby miał więcej mocy. Jądro liczy tę wartość w sposób ciągły i uśrednia ją wykładniczo za trzy okresy: ostatnią minutę, ostatnie 5 minut i ostatnie 15 minut. Liczba sama w sobie nic nie mówi — trzeba ją zestawić z liczbą rdzeni procesora (`nproc`), bo load 4 na jednym rdzeniu to katastrofa, a na szesnastu to nic.

```
top - 14:32:07 up 23 days,  4:12,  2 users,  load average: 3.41, 2.87, 1.02
Tasks: 187 total,   2 running, 184 sleeping,   0 stopped,   1 zombie
%Cpu(s): 22.1 us,  4.3 sy,  0.0 ni, 58.2 id, 14.8 wa,  0.0 hi,  0.6 si,  0.0 st
MiB Mem :  15872.0 total,   1203.4 free,   9821.7 used,   4846.9 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   5432.1 avail Mem
```

- **`load average: 3.41, 2.87, 1.02`** — trzy wartości to kolejno średnia za 1, 5 i 15 minut. Trend rosnący od 1.02 (15 min temu) do 3.41 (teraz) mówi więcej niż sama wartość bieżąca — system dopiero się obciąża, a nie właśnie wychodzi z obciążenia.
- **`Tasks`** — liczba procesów wg stanu. Jedna pozycja `zombie` to nic, sto pozycji `zombie` to błąd w aplikacji (pełny temat: następna strona rozdziału).
- **`%Cpu(s)`** — rozbicie czasu CPU:
- `us` (user) — czas w programach użytkownika,
- `sy` (system) — czas w jądrze (wywołania systemowe, obsługa przerwań),
- `ni` (nice) — czas procesów z niestandardowym priorytetem,
- `id` (idle) — czas bezczynności,
- **`wa` (I/O wait)** — CPU czeka, bo proces czeka na dysk/sieć. Wysokie `wa` przy niskim `us` to sygnał: problem nie jest w procesorze, jest w podsystemie dyskowym.
- `hi`/`si` — czas obsługi przerwań sprzętowych/programowych,
- `st` (steal) — czas skradziony przez hypervisor, patrz ramka niżej.
- **`buff/cache`** — pamięć zajęta przez bufory i cache strony. To nie jest „zajęta" pamięć w sensie alarmowym — jądro odda ją natychmiast, gdy aplikacja jej zażąda. Duże `buff/cache` przy małym `free` to normalny, zdrowy Linux, nie brak pamięci.

> **Pojęcie: przełączanie kontekstu.** Jeden fizyczny rdzeń wykonuje w danej chwili instrukcje tylko jednego procesu — wrażenie, że dziesiątki procesów działają „naraz", bierze się z tego, że jądro w rytmie milisekund zatrzymuje bieżący proces, zapisuje jego rejestry i stan, i podmienia go na inny z kolejki gotowych do działania. Taka podmiana nazywa się przełączeniem kontekstu (context switch) i sama w sobie kosztuje czas procesora — zapisanie i odtworzenie stanu, przeładowanie pamięci podręcznej. Kolumna `sy` w `%Cpu(s)` rośnie właśnie wtedy, gdy system robi dużo takiej pracy administracyjnej zamiast liczyć dla aplikacji, a liczbę przełączeń na sekundę pokazuje `vmstat 1` w kolumnie `cs`.

> **Pojęcie: czas skradziony (steal time).** Na maszynie wirtualnej fizyczny procesor jest dzielony między kilka VM-ów przez hypervisor, który przydziela im czas CPU w kawałkach, tak jak jądro przydziela czas procesom. Steal time to procent czasu, w którym Twoja maszyna wirtualna *chciała* wykonywać instrukcje, ale hypervisor akurat oddał fizyczny rdzeń innej maszynie na tym samym hoście. To jedyna kolumna w `%Cpu(s)`, której przyczyny nie da się naprawić z poziomu Twojego systemu — wysoki `st`

(kilka procent i więcej) to sygnał, że problem leży w warstwie wirtualizacji albo w przewymiarowaniu (overcommit) zasobów przez dostawcę, a nie w Twojej aplikacji.

Krok po kroku

1. Odczytaj load average względem liczby rdzeni.

```
nproc
```

```
4
```

Load average 3.41 na czterordzeniowej maszynie to realne obciążenie bliskie granicy — każdy rdzeń ma niemal ciągle coś do roboty. Ta sama wartość na maszynie 32-rdzeniowej to 10% wykorzystania, nic niepokojącego.

2. Znajdź źródło wysokiego `wa`.

Naciśnij `1` w `top`, żeby rozbić `%Cpu(s)` na rdzenie — czasem jeden rdzeń jest zablokowany, a reszta próżnuje, co ginie w uśrednionej wartości zbiorczej. Potem posortuj wg stanu procesów w `D` (uninterruptible sleep, patrz następna strona) albo użyj `iostat -x 1` (pakiet `sysstat`), żeby zobaczyć, który dysk jest przeciążony.

3. Uruchom `top` w trybie wsadowym do skryptu.

```
top -b -n 1 -o %MEM | head -17
```

```
top - 14:40:02 up 23 days,  4:20,  2 users,  load average: 2.10, 2.55, 1.80
...
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1203 postgres  20   0 2481200 943212  81204 S   1.3   5.9   3:12.44 postgres
```

`-b` (batch) wyłącza interfejs interaktywny i drukuje zwykły tekst, `-n 1` każe zrobić dokładnie jedno odświeżenie i zakończyć. To jest wersja `top`, którą wrzucasz do crona albo do skryptu diagnostycznego — bez `-n` `top -b` leciałby w nieskończoność, zapychając log.

4. Zainstaluj i uruchom `htop`.

Na RHEL/Rocky 9 i 10 `htop` nie jest częścią BaseOS ani AppStream — trzeba włączyć EPEL:

```
sudo dnf install -y epel-release
sudo dnf install -y htop
```

Na Debianie/Ubuntu jest w domyślnych repozytoriach:

```
sudo apt install htop
```

Na SLES `htop` leży w repozytorium Packman albo w dodatku PackageHub — domyślne repozytoria SLE go nie zawierają.

5. Filtruj i zabijaj w `htop`.

Uruchom `htop`, naciśnij `F4`, wpisz fragment nazwy (np. `nginx`) — lista zawęzi się do pasujących procesów w czasie rzeczywistym, bez zamykania i ponownego uruchamiania. Zaznacz kilka procesów spacją, naciśnij `F9`, wybierz `SIGTERM` (albo `SIGKILL`, jeśli proces nie reaguje) — sygnał poleci do wszystkich zaznaczonych naraz, bez ręcznego wpisywania każdego PID-u osobno. To jest największa praktyczna przewaga `htop` nad `top`: `top` zabija jeden proces na raz i wymaga wpisania PID-u z klawiatury.

6. Sprawdź priorytet i popraw go.

```
ps -o pid,ni,cmd -p 1234
renice -n 10 -p 1234
```

```
    PID    NI CMD
   1234     0 /usr/bin/backup-script.sh


1234 (process ID) old priority 0, new priority 10
```

Zakres `nice` to `-20` (najwyższy priorytet, tylko root) do `19` (najniższy). Proces w tle, który nie musi być responsywny (kompresja, backup), warto uruchamiać z dodatnim `nice`, żeby ustępował miejsca procesom interaktywnym. `ionice -c 3` robi to samo dla kolejki I/O — klasa `3` (idle) oddaje pierwszeństwo dyskowe każdemu innemu procesowi.

Częste błędy

Panika na widok load average bez sprawdzenia liczby rdzeni.

```
load average: 8.02, 7.55, 6.90
```

Administrator widzi „8" i uznaje serwer za przeciążony. Maszyna ma 16 rdzeni — realne wykorzystanie to 50%, komfortowy zapas. Zawsze zestawiaj load average z `nproc`, nigdy nie

oceniaj go w oderwaniu.

Ignorowanie `st` na maszynie wirtualnej.

Proces zjada rzekomo „100% CPU" według `%CPU` w kolumnie procesu, ale suma `us+sy` w nagłówku to tylko 40%, a `st` pokazuje 55%. Ktoś próbuje zoptymalizować kod aplikacji, podczas gdy prawdziwym problemem jest przeciążony hypervisor, który oddaje Twojej VM za mało czasu procesora. Rozwiązanie nie leży w Twoim systemie — trzeba zgłosić to zespołowi odpowiedzialnemu za platformę wirtualizacji albo hosting.

`top -p` z listą PID-ów, która nie łapie nowych procesów.

```
top -p "$(pgrep -d, -f myapp)"
```

Aplikacja restartuje się z nowym PID-em (np. po `systemctl restart`), a `top` dalej pokazuje stary, już nieistniejący proces (albo pustą listę). `-p` zamraża listę PID-ów w momencie startu `top` — nie dopisuje nowych automatycznie. Trzeba wyjść (`q`) i uruchomić `top` ponownie z odświeżoną listą, albo śledzić po nazwie przez `htop` z `F4`, które filtruje dynamicznie.

Zabicie procesu z `top` bez sprawdzenia, co to za proces.

Naciskasz `k`, wpisujesz PID z pamięci sprzed chwili, potwierdzasz sygnał — i okazuje się, że lista się przesortowała między spojrzeniem a naciśnięciem klawisza, więc zabiłeś inny proces niż zamierzałeś. Zanim naciśniesz `k`, zatrzymaj odświeżanie klawiszem spacji (pauza) albo zweryfikuj PID przez `ps -p PID -o cmd=` tuż przed wysłaniem sygnału.

Traktowanie `buff/cache` jako brakującej pamięci.

```
MiB Mem :  15872.0 total,    340.2 free,   4200.1 used,  11331.7 buff/cache
```

Widząc 340 MB w kolumnie `free`, ktoś zamawia więcej RAM-u dla serwera, który wcale go nie potrzebuje. `buff/cache` to pamięć, którą jądro oddaje aplikacjom natychmiast na żądanie — to nie jest zużycie w sensie alarmowym, tylko efektywne wykorzystanie wolnej pamięci pod cache dysku. Prawdziwy wskaźnik presji pamięciowej to `avail Mem` w tej samej linii (tu: realnie dostępne 15,5 GB) albo obecność aktywnego swapu.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, w katalogu `/tmp`. Nic nie wymaga uprawnień administratora.

Ćwiczenie 1 — priorytet CPU pod obciążeniem

```
cd /tmp

# 1. Wygeneruj obciążenie CPU na chwilę
yes > /dev/null &
PID_CPU=$!

# 2. Zobacz je w top
top -b -n 1 | head -12
```

```
top - 14:55:11 up 1 day,  2:03,  1 user,  load average: 1.20, 0.45, 0.20
...
%Cpu(s): 98.2 us,  1.1 sy,  0.0 ni,  0.0 id,  0.0 wa, ...
```

Zwróć uwagę, że `us` skoczyło blisko 100% — cały czas idzie w kod użytkownika, `yes` nie robi żadnych wywołań systemowych poza zapisem do `/dev/null`.

```
# 3. Zmień priorytet działającego procesu i sprawdź efekt
renice -n 15 -p $PID_CPU
ps -o pid,ni,pcpu -p $PID_CPU

# 4. Posprzątaj
kill $PID_CPU
```

Co pokaże kolumna `NI` po `renice` i kiedy zmiana priorytetu w ogóle wpływa na `%CPU`?

Odpowiedź

Po `renice` kolumna `NI` pokaże `15`. Na maszynie jednordzeniowej z jednym konkurującym procesem `%CPU` może się nie zmienić wcale — priorytet ma znaczenie dopiero, gdy o CPU rywalizuje kilka procesów naraz. Odpal drugi `yes > /dev/null &` bez zmiany priorytetu i porównaj, który z dwóch dostaje więcej czasu procesora w `top -b -n 1`.

Jeśli masz zainstalowany `htop`: uruchom go, naciśnij `F4`, wpisz `yes`, zaznacz proces spacją i wyślij `SIGTERM` przez `F9` — bez wychodzenia z ekranu i bez ręcznego wpisywania PID-u.

Pytania kontrolne

1. Load average pokazuje `12.40, 11.80, 9.02` na maszynie z `nproc` równym `16`. Czy to problem?

Odpowiedź

Nie od razu. 12,4 na 16 rdzeniach to ok. 78% wykorzystania — wysokie, ale nie krytyczne, i rosnące od 9,02 (15 min temu), co sugeruje narastające obciążenie, nie chwilowy skok. Samą liczbę zawsze trzeba dzielić przez liczbę rdzeni, żeby miała sens.

2. `%Cpu(s)` pokazuje `us: 12.0`, `wa: 61.4`. Gdzie szukać przyczyny, w kodzie aplikacji czy gdzie indziej?

Odpowiedź

Poza kodem aplikacji. Wysoki `wa` przy niskim `us` znaczy, że procesor w większości czasu nic nie liczy, tylko czeka na zakończenie operacji I/O — dysk, sieciowy system plików, wolne repliki bazy danych. Optymalizacja kodu aplikacji nic tu nie da; trzeba zdiagnozować podsystem I/O (`iostat -x 1`, sprawdzenie procesów w stanie `D`).

3. Na maszynie wirtualnej `st` w nagłówku `top` wynosi `18.5`. Co to znaczy i kto powinien to naprawić?

Odpowiedź

`st` to procent czasu procesora, który Twoja VM chciała wykorzystać, ale fizyczny CPU był w tym czasie przydzielony innej maszynie wirtualnej na tym samym hoście. To sygnał przeciążenia hosta wirtualizacji albo zbyt agresywnego przewymiarowania (overcommit) zasobów przez dostawcę. Naprawa leży poza Twoim systemem — zgłoszenie do zespołu platformy albo dostawcy chmury, nie zmiana konfiguracji w VM.

4. Dlaczego `top -p 1234,5678` po restarcie jednej z monitorowanych usług przestaje ją pokazywać, mimo że usługa działa?

Odpowiedź

`top -p` zamraża listę PID-ów w momencie uruchomienia. Restart usługi (np. przez `systemctl restart`) kończy stary proces i uruchamia nowy z innym PID-em — `top` dalej pilnuje starego, już nieistniejącego numeru. Trzeba wyjść i uruchomić `top` z nową listą PID-ów, albo monitorować

po nazwie procesu przez `htop` z filtrem `F4`, który odświeża dopasowanie na bieżąco.

Powiązane: ps: zdjęcie stanu procesów · Stany procesów: zombie, sieroty, D-state · Sygnały: kill, pkill, killall, trap

Źródła: `man 1 top` · `man 1 htop` · `man 1 renice` · `man 1 ionice` · `man 1 pidstat` · Red Hat: How to use EPEL · EPEL na Rocky/RHEL 9-10 — LinuxCapable

---

Revision #18
Created 2026-09-17 15:07:12 UTC by Tomasz Konieczny
Updated 2026-10-03 19:08:11 UTC by Tomasz Konieczny