ps: zdjęcie stanu procesów

◫◫ Poziom: podstawy

◫◫ Dotyczy: RHEL/Rocky 9–10 (procps-ng), różnice Debian/Ubuntu i SLES zaznaczone

▯ Czas: ~16 min

`ps` robi zdjęcie stanu procesów w danej chwili — nie monitoruje niczego, tylko odczytuje `/proc` raz i kończy pracę. Ta strona to komplet: składnia, kolumny, kody `STAT` i miejsce, z którego `ps` w ogóle bierze dane.

◫◫ Skróty użyte na tej stronie (13)

Skrót	Co znaczy
BSD	Berkeley Software Distribution: rodzina Uniksów z uczelni Berkeley, inna gałąź niż Linux, stąd czasem inna składnia tych samych poleceń
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
JVM	Java Virtual Machine: maszyna wirtualna, w której działają programy w Javie
PID	Process ID: numer procesu w systemie
POSIX	Portable Operating System Interface: wspólny zestaw zasad, dzięki któremu polecenie działa tak samo na różnych Uniksach
PPID	Parent Process ID: numer procesu rodzica, czyli tego, który uruchomił dany proces
RAM	Random Access Memory: pamięć operacyjna
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
RSS	Resident Set Size: ile pamięci RAM proces faktycznie zajmuje w tej chwili, w odróżnieniu od VSZ
SID	w Oracle: System Identifier, nazwa instancji; w Windows: Security Identifier, numer konta
SLES	SUSE Linux Enterprise Server: płatna dystrybucja Linuksa firmy SUSE, konkurent Rocky/RHEL
UID	User ID: numer użytkownika w Linuksie
VSZ	Virtual Size: cała pamięć zaadresowana przez proces, także ta niewykorzystana, w kolumnie ps zwykle dużo większa niż RSS

Fiszka

◫◫ TYLKO ODCZYT

Wypisanie procesów bieżącego terminala

```
ps
```

Bez żadnych flag `ps` pokazuje wyłącznie procesy uruchomione w bieżącym terminalu i należące do użytkownika, który go wywołał — najwęższy możliwy widok. Codzienną pracę zaczynamy od wariantów niżej, które pokazują cały system.

Przykład:

```
  PID TTY          TIME CMD
 4502 pts/0    00:00:00 bash
 4600 pts/0    00:00:00 ps
```

◫◫ **ps** = process status: pokaż procesy

◫◫ TYLKO ODCZYT

Wypisanie wszystkich procesów: `aux`

```
ps aux | head -5
```

Klasyk administratora: wszystkie procesy w systemie, wszyscy użytkownicy, także demony bez przypisanego terminala. Litery nie mają myślnika — to składnia BSD.

Przykład:

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1 171004 11840 ?        Ss   wrz16   0:03
/usr/lib/systemd/systemd --switched-root --system --deserialize=31
root         2  0.0  0.0      0     0 ?        S    wrz16   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   wrz16   0:00 [rcu_gp]
adm_user  4821  0.2  0.4  55600  8420 ?        Ss   09:12   0:03 nginx: master process
/usr/sbin/nginx
```

◫◫ **BSD** = Berkeley Software Distribution: rodzina Uniksów z uczelni Berkeley, inna gałąź niż Linux, stąd czasem inna składnia tych samych poleceń · **ps** = process status: pokaż procesy

◫◫ TYLKO ODCZYT

Wypisanie wszystkich procesów z PPID: `-ef`

```
ps -ef | head -5
```

To samo pokrycie co `aux`, ale w tradycji UNIX/POSIX — z myślnikiem, z kolumną `PPID` zamiast `%CPU`/`%MEM`. Sięgamy po to, gdy interesuje nas drzewo rodzic–dziecko, nie zużycie zasobów.

Przykład:

```
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 wrz16 ?        00:00:03 /usr/lib/systemd/systemd --switched-
root --system --deserialize=31
root           2       0  0 wrz16 ?        00:00:00 [kthreadd]
root           3       2  0 wrz16 ?        00:00:00 [rcu_gp]
adm_user    4821       1  0 09:12 ?        00:00:03 nginx: master process
/usr/sbin/nginx
```

PPID = Parent Process ID: numer procesu rodzica, czyli tego, który uruchomił dany proces · **CPU** = Central Processing Unit: procesor · **ps** = process status: pokaż procesy

TYLKO ODCZYT

Filtrowanie po konkretnych PID-ach: `-p`

```
ps -p 4821,4822
```

Zamiast przeglądać całą listę i szukać wzrokiem, podajemy numery, które już znamy — z `pgrep`, z `pidof` albo z poprzedniego polecenia.

Przykład:

```
    PID TTY          TIME CMD
   4821 ?        00:00:03 nginx
   4822 ?        00:00:00 nginx
```

ps = process status: pokaż procesy

TYLKO ODCZYT

Filtrowanie po właścicielu: `-u`

```
ps -u nazwa_usera
```

Pokazujemy tylko procesy należące do wskazanego użytkownika, niezależnie od terminala. `-U` robi to samo dla właściciela *rzeczywistego* (real UID) — różnica ma znaczenie przy procesach, które zmieniły tożsamość przez `setuid`.

Przykład:

```
  PID TTY          TIME CMD
 4502 pts/0    00:00:00 bash
 6011 ?        00:00:00 sshd
```

UID = User ID: numer użytkownika w Linuksie · **ps** = process status: pokaż procesy

TYLKO ODCZYT

Filtrowanie po nazwie polecenia: `-C`

```
ps -C sshd
```

Filtrujemy po polu `comm` — dokładnej nazwie programu, bez ścieżki i argumentów. Szybszy sposób na znalezienie konkretnego demona niż `ps aux | grep`.

Przykład:

```
  PID TTY          TIME CMD
  900 ?        00:00:00 sshd
```

ps = process status: pokaż procesy

TYLKO ODCZYT

Wybór własnego zestawu kolumn: `-o`

```
ps -o pid,ppid,stat,etime,rss,comm -p 4821
```

Zamiast zgadywać, co znaczy litera w `aux`, podajemy dokładnie te kolumny, które nas interesują. Nazwy pól rozdzielamy przecinkiem, bez spacji — spacja rozdzieliłaby to na dwa osobne argumenty.

Przykład:

```
  PID   PPID STAT     ELAPSED   RSS COMMAND
 4821      1 Ss          02:14  8420 nginx
```

ps = process status: pokaż procesy

TYLKO ODCZYT

Sortowanie wyniku: `--sort`

```
ps aux --sort=-%mem | head -5
```

Minus przed nazwą pola sortuje malejąco. W `-o` używamy nazwy podanej kolumny (`--sort=-rss`), w gotowym `aux` mamy dostęp do `%cpu`, `%mem` i `vsz`.

Przykład:

```
USER       PID %CPU %MEM    VSZ    RSS TTY      STAT START   TIME COMMAND
postgres  1203  1.8  5.9 2481200 943212 ?      Ss   09:10   3:12 postgres: writer
process
root         1  0.0  0.1  171004  11840 ?      Ss   wrz16   0:03
/usr/lib/systemd/systemd
adm_user  4821  0.3  0.4   55600   8420 ?      S    09:12   0:03 nginx: master process
```

ps = process status: pokaż procesy

TYLKO ODCZYT

Wypisanie wątków zamiast procesów: `-eLf`

```
ps -eLf | awk 'NR==1 || $4==1203'
```

`-L` dokłada kolumnę `LWP` i pokazuje każdy wątek procesu jako osobny wiersz. Bez tej flagi wielowątkowy proces — JVM, baza danych — wygląda na liście jak jeden wiersz, niezależnie od tego, ile ma faktycznie wątków.

Przykład:

```
UID          PID    PPID     LWP  C NLWP STIME TTY          TIME CMD
postgres    1203       1    1203  0   12 09:10 ?        00:00:01 postgres: writer
postgres    1203       1    1207  0   12 09:10 ?        00:00:03 postgres: writer
```

JVM = Java Virtual Machine: maszyna wirtualna, w której działają programy w Javie · **ps** = process status: pokaż procesy

TYLKO ODCZYT

Wypisanie drzewa procesów: `auxf`

```
ps auxf | head -8
```

`f` (forest) dorysowuje kreski `\_`, które pokazują, kto jest czyim dzieckiem, bez wychodzenia z `ps` po dodatkowe narzędzie. Równoważne: `ps -ef --forest`.

Przykład:

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1 171004 11840 ?        Ss   wrz16   0:03
/usr/lib/systemd/systemd
root      1234  0.0  0.2  55600  9820 ?        Ss   09:12   0:00 /usr/sbin/httpd
apache    1240  0.0  0.1  55600  5210 ?        S    09:12   0:02  \_ /usr/sbin/httpd
apache    1241  0.0  0.1  55600  5180 ?        S    09:12   0:01  \_ /usr/sbin/httpd
```

ps = process status: pokaż procesy

TYLKO ODCZYT

Wypisanie drzewa procesów bez `ps`: `pstree`

```
pstree -p | head
```

Osobny pakiet (`psmisc` — tak samo na RHEL/Rocky, Debianie/Ubuntu i SLES), który rysuje wyłącznie drzewo nazw, bez kolumn liczbowych. `-p` dopisuje numer PID w nawiasie przy każdej nazwie.

Przykład:

```
systemd(1)-+-NetworkManager(812)-+-{NetworkManager}(813)
           |                     `-{NetworkManager}(814)
           |-agetty(945)
           |-auditd(780)-+-{auditd}(781)
           |             `-{auditd}(782)
           |-chronyd(772)
```

```
           |-crond(801)
           |-dbus-broker(790)---dbus-broker-lau(789)
           |-httpd(1234)-+-httpd(1240)
           |             `-httpd(1241)
```

RHEL = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **SLES** = SUSE Linux Enterprise Server: płatna dystrybucja Linuksa firmy SUSE, konkurent Rocky/RHEL · **PID** = Process ID: numer procesu w systemie

TYLKO ODCZYT

Szukanie procesu po nazwie bez łapania samego siebie: `pgrep -a`

```
pgrep -a nginx
```

Filtrujemy procesy jądrowo, bez przechodzenia przez `grep` w ogóle — więc problem złapania samego siebie na liście wyników (patrz *Częste błędy*) tu nie istnieje. `-a` dopisuje pełną linię poleceń obok PID-u.

Przykład:

```
4821 nginx: worker process
4822 nginx: worker process
```

pgrep = process grep: znajdź numer procesu po nazwie bez przeglądania pełnej listy ps

TYLKO ODCZYT

Odczytanie pełnej linii poleceń: `/proc/PID/cmdline`

```
cat /proc/4821/cmdline | tr '\0' ' '; echo
```

Argumenty w tym pliku są rozdzielone bajtem `\0`, nie spacją — stąd `tr '\0' ' '`, żeby dało się to w ogóle przeczytać na ekranie.

Przykład:

```
nginx: master process /usr/sbin/nginx
```

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Odczytanie stanu procesu z jądra: `/proc/PID/status`

```
cat /proc/4821/status | head
```

Surowe dane, z których `ps` składa swoje kolumny — `State` odpowiada `STAT`, `VmRSS` odpowiada `RSS`, `PPid` odpowiada `PPID`. Przydaje się, gdy chcemy sprawdzić coś, czego żaden format `-o` nie pokazuje wprost.

Przykład:

```
Name:	nginx
Umask:	0022
State:	S (sleeping)
Tgid:	4821
Ngid:	0
Pid:	4821
PPid:	1
TracerPid:	0
Uid:	0	0	0	0
Gid:	0	0	0	0
```

RSS = Resident Set Size: ile pamięci RAM proces faktycznie zajmuje w tej chwili, w odróżnieniu od VSZ · **PPID** = Parent Process ID: numer procesu rodzica, czyli tego, który uruchomił dany proces · **cat** = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Sprawdzenie, jaką binarkę proces faktycznie uruchomił: `/proc/PID/exe`

```
readlink /proc/4821/exe
```

Dowiązanie do binarki, którą proces uruchomił przy starcie — nawet jeśli plik na dysku został od tego czasu podmieniony albo usunięty. W tym drugim wypadku dowiązanie ma dopisek `(deleted)`, klasyczny sygnał, że usługę trzeba zrestartować po aktualizacji pakietu.

Przykład:

```
/usr/sbin/nginx
```

readlink = pokaż, na co wskazuje dowiązanie symboliczne

TYLKO ODCZYT

Sprawdzenie katalogu roboczego procesu: `/proc/PID/cwd`

```
readlink /proc/4821/cwd
```

Katalog, z którego proces został uruchomiony i względem którego rozwiązuje ścieżki względne. Przydaje się, gdy proces otwiera plik logu przez ścieżkę względną i nie wiadomo, gdzie go szukać.

Przykład:

```
/
```

readlink = pokaż, na co wskazuje dowiązanie symboliczne

TYLKO ODCZYT

Wypisanie otwartych deskryptorów plików: `/proc/PID/fd`

```
ls -l /proc/4821/fd
```

Każdy wpis to jeden otwarty plik, gniazdo albo potok. Numer to sam deskryptor, cel dowiązania mówi, co za nim stoi — plik na dysku, gniazdo sieciowe, urządzenie.

Przykład:

```
total 0
lrwx------ 1 root root 64 wrz 17 09:12 0 -> /dev/null
lrwx------ 1 root root 64 wrz 17 09:12 1 -> /dev/null
lrwx------ 1 root root 64 wrz 17 09:12 2 -> /dev/null
lrwx------ 1 root root 64 wrz 17 09:12 6 -> socket:[28841]
lrwx------ 1 root root 64 wrz 17 09:12 7 -> anon_inode:[eventfd]
```

ls = list: wypisz zawartość katalogu

TYLKO ODCZYT

Wszystkie flagi

Flaga / kolumna	Co robi	Kiedy używasz
`ps` (bez flag)	tylko procesy Twojego terminala	rzadko — zwykle chcesz szerszy widok
`ps aux`	BSD-owy zestaw: wszystkie procesy, wszyscy użytkownicy, także bez terminala	domyślny odruch administratora
`ps -ef`	UNIX-owy zestaw: wszystkie procesy z PPID zamiast %CPU/%MEM	gdy interesuje Cię drzewo rodzic-dziecko
`-p PID,...`	filtr po konkretnych PID-ach	znasz już numer, chcesz tylko jego szczegóły
`-u user`, `-U user`	filtr po właścicielu efektywnym / rzeczywistym	zawężasz do procesów jednej osoby
`-C nazwa`	filtr po nazwie polecenia (`comm`)	szukasz konkretnego demona po nazwie binarki
`-o pola`	własny, dowolny zestaw kolumn	gdy `aux`/`-ef` nie mają tego, czego szukasz
`--sort=-pole`	sortuje malejąco po polu z `-o` lub `-%cpu`/`-%mem`/`-vsz` w `aux`	szukasz hoga zasobów
`auxf` / `-ef --forest`	drzewo procesów w samym `ps`	podejrzewasz, że coś odpaliło dziwne dziecko
`-eLf`	rozbija procesy wielowątkowe na wątki (LWP)	JVM, bazy danych — jeden proces, setki wątków

Uważaj

- `ps aux` i `ps -aux` to formalnie różne polecenia — modyfikator opisany w sekcji *Krok po kroku*.
- Suma kolumny RSS ze wszystkich procesów **przekracza** ilość fizycznego RAM-u — biblioteki współdzielone liczą się osobno w każdym procesie.
- `ps aux | grep nazwa` złapie też proces samego `grep`, jeśli nie użyjesz triku `[n]azwa` albo `pgrep`.

PPID = Parent Process ID: numer procesu rodzica, czyli tego, który uruchomił dany proces · **CPU** = Central Processing Unit: procesor · **PID** = Process ID: numer procesu w systemie · **JVM** = Java Virtual Machine: maszyna wirtualna, w której działają programy w Javie · **RSS** = Resident Set Size: ile pamięci RAM proces faktycznie zajmuje w tej chwili, w odróżnieniu od VSZ

Z życia wzięte

Z ŻYCIA WZIĘTE

O 9:12 pg01 zaczyna zwalniać, a Ola podejrzewa, że winny jest jeden z procesów postgresa. Jednym poleceniem ps aux --sort=-%mem sprawdzamy, który proces trzyma najwięcej RAM-u, i widzimy zapomniany raport, uruchomiony ręcznie tydzień temu i wciąż czekający w pamięci. Tomek go kończy, a pg01 wraca do normalnego zużycia.

Po co to istnieje

Pojęcie: proces. Proces to uruchomiony program razem z całym stanem, jaki system mu przydzielił w danej chwili: własną przestrzenią pamięci, zestawem otwartych plików, katalogiem roboczym, użytkownikiem, na którego prawach działa, i unikalnym numerem identyfikacyjnym zwanym PID. Sam plik na dysku — na przykład `/usr/sbin/nginx` — to tylko martwy ciąg bajtów; dopiero jego uruchomienie tworzy proces, który jądro zaczyna śledzić, planować na procesorze i rozliczać z zasobów. Ten sam plik wykonywalny uruchomiony trzy razy to trzy zupełnie osobne procesy, każdy z własnym PID-em, własną pamięcią i własnym stanem — nawet jeśli wszystkie trzy wykonują dokładnie ten sam kod. `ps`, `top` i cały ten rozdział dotyczą właśnie zarządzania tymi obiektami: jak je zobaczyć, jak ocenić ich stan i jak bezpiecznie nimi sterować.

Chcesz wiedzieć, co się dzieje w systemie *teraz*: ile jest procesów, kto je uruchomił, ile zżera pamięci, czy któryś się zawiesił. `ps` (process status) odpowiada na to pytanie tak, jak odpowiada się na każde pytanie o stan w Uniksie — czytając plik. W tym wypadku pseudo-plikowy system `/proc`, który jądro wystawia właśnie po to, żeby narzędzia userspace mogły zaglądać do struktur procesów bez własnych uprawnień jądra.

`ps` istnieje od pierwszych wersji Uniksa i przez dekady rozjechał się na dwie tradycje — BSD i System V — które dały różne składnie tego samego polecenia. GNU/Linux (a więc i dzisiejszy `procps-ng`, standard na RHEL, Debianie i SLES) połączył obie tradycje w jednym binarce, co jest wygodne i jednocześnie źródłem większości nieporozumień opisanych niżej.

Analogia

`ps` to **zdjęcie grupowe hali produkcyjnej** zrobione o 14:03:17.

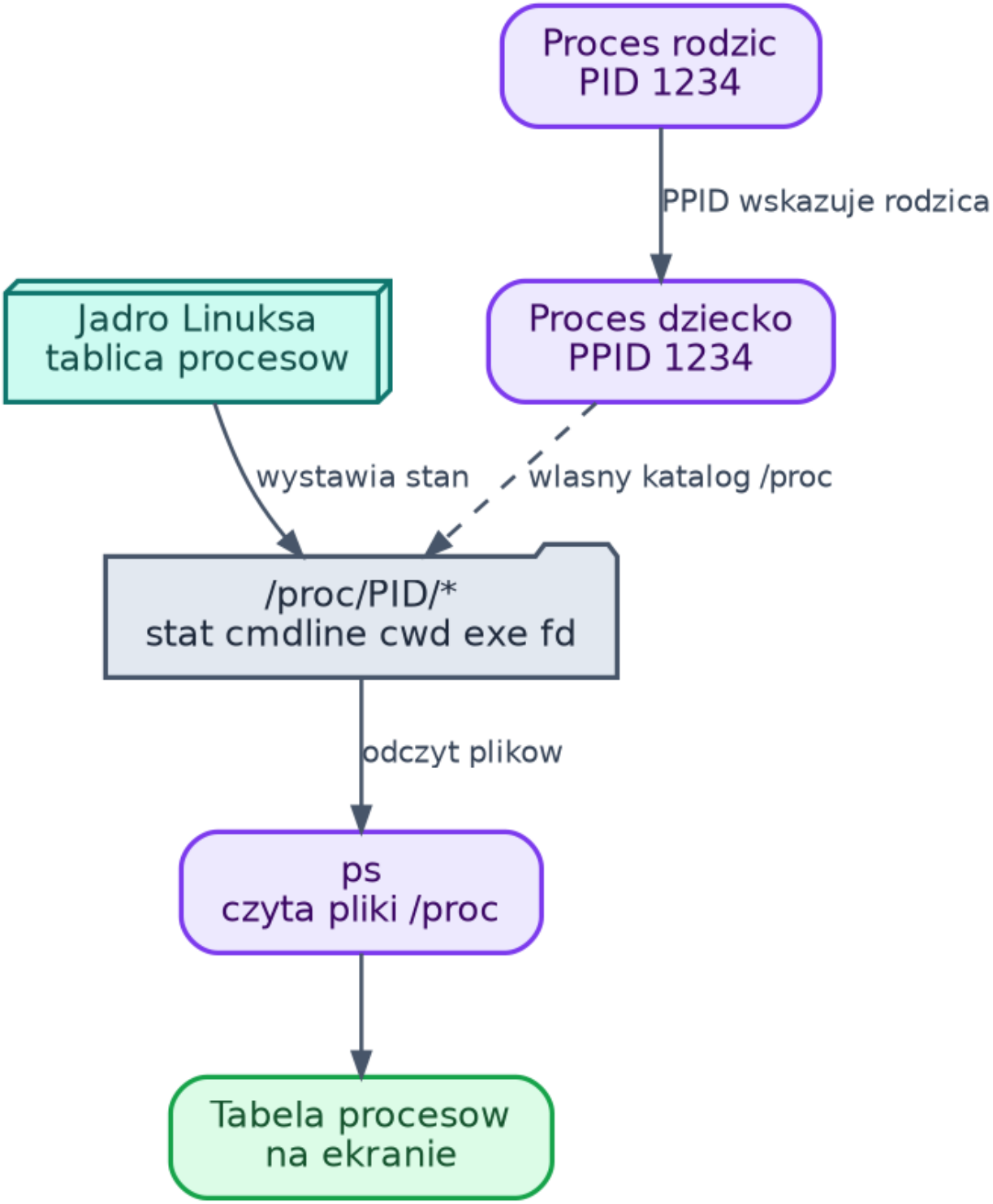
Fotograf wchodzi na halę, robi jedno ujęcie i wychodzi. Na zdjęciu widzisz, kto gdzie stoi, kto trzyma narzędzie, kto siedzi przy biurku. Nie widzisz, czy osoba przy taśmie właśnie biegnie, czy stoi bez ruchu od godziny — zdjęcie zamraża jedną chwilę, nie ruch. Jeśli chcesz zobaczyć ruch, potrzebujesz kamery, nie aparatu — i dokładnie to robi `top` (kolejna strona w tym rozdziale).

Zdjęcie ma jeszcze jedną właściwość, o której administratorzy zapominają: samo zrobienie zdjęcia też jest zdarzeniem na hali. Proces `ps` sam pojawia się na liście procesów, które właśnie fotografuje — o czym przekonasz się boleśnie, próbując grepować po nazwie procesu (patrz *Częste błędy*). Fotograf wchodzi w kadr własnego zdjęcia.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: PID i PPID.** Każdy proces w systemie dostaje przy starcie unikalny numer identyfikacyjny — PID (process ID) — który nie powtórzy się ponownie, dopóki proces żyje, ani przez chwilę zaraz po jego śmierci. PPID (parent PID) to PID procesu, który dany proces utworzył — każdy proces w Linuksie, poza PID-em 1, ma w danej chwili dokładnie jednego rodzica. Kolumna PPID jest tym, co pozwala złożyć płaską listę z `ps` w drzewo: proces o PPID równym 1234 jest dzieckiem procesu 1234.



Jadro wystawia stan kazdego procesu jako katalog /proc/PID, ps go odczytuje i skleja w tabele, a PPID pozwala odtworzyc drzewo rodzic-dziecko (kliknięcie otwiera pełny rozmiar)

Jądro Linuksa udostępnia stan każdego procesu jako katalog `/proc/PID/`. W środku leżą m.in.:

- `/proc/PID/stat` i `/proc/PID/status` — stan, PPID, zużycie pamięci, priorytet,
- `/proc/PID/cmdline` — pełna linia poleceń, z argumentami rozdzielonymi bajtem `\0` (nie spacją — stąd `tr '\0' ' '` w Fiszce),
- `/proc/PID/environ` — zmienne środowiskowe procesu, też rozdzielone `\0`,
- `/proc/PID/cwd` — dowiązanie symboliczne do katalogu roboczego procesu,
- `/proc/PID/exe` — dowiązanie do binarki, którą proces faktycznie uruchomił (przydatne, gdy plik na dysku został podmieniony po starcie),
- `/proc/PID/fd/` — katalog z otwartymi deskryptorami plików procesu.

`ps` po prostu odczytuje te pliki dla każdego katalogu `/proc/<liczba>/`, składa z nich tabelkę i kończy działanie. Nie ma tu żadnej magii ani demona w tle — to jest powód, dla którego `ps` jest tak tani w wywołaniu i dlaczego liczby, które widzisz, są prawdziwe na moment odczytu, a nie uśrednione w czasie.

> **Pojęcie: pamięć wirtualna kontra rezydentna (VSZ/RSS).** VSZ (virtual size) to cała przestrzeń adresowa, którą proces *może* zaadresować: kod, dane, stos, biblioteki współdzielone, także strony, które nigdy nie zostały fizycznie użyte albo zostały wyrzucone do swapu. RSS (resident set size) to ta część, która faktycznie leży w RAM-ie w tej chwili. RSS jest zawsze podzbiorem VSZ — odwrotna sytuacja nie zdarza się nigdy.

Proces może mieć VSZ 4 GB i RSS 30 MB — to normalne dla procesu, który zamapował duży plik, ale dotknął tylko jego skrawka. Liczba VSZ jest często nabita przez `mmap()` dużych plików czy zarezerwowaną (ale nietkniętą) pamięć, więc sama w sobie niewiele mówi o realnym zużyciu.

Pułapka w drugą stronę: zsumowanie RSS wszystkich procesów **nie da** realnego zużycia RAM-u. Biblioteka `libc.so` jest zamapowana do pamięci fizycznej raz, ale każdy proces, który jej używa, wykazuje ją w swoim RSS. Zsumuj RSS stu procesów korzystających z tej samej biblioteki, a dostaniesz sto razy jej rozmiar doliczone do realnego zużycia. Do prawdziwego obrazu pamięci systemu służy `free -h` albo, dla pojedynczego procesu, `PSS` z `/proc/PID/smaps_rollup` (Proportional Set Size — dzieli strony współdzielone przez liczbę procesów, które ich używają).

Kolumna STAT. Pierwsza litera to stan procesu, kolejne znaki to modyfikatory:

Kod	Znaczenie
`R`	running — działa albo czeka w kolejce na CPU (runnable)
`S`	interruptible sleep — czeka na zdarzenie (np. wejście z sieci), reaguje na sygnały
`D`	uninterruptible sleep — czeka na I/O jądra, **nie reaguje nawet na SIGKILL** (pełny opis: następna strona rozdziału)

Kod	Znaczenie
`T`	zatrzymany sygnałem (`SIGSTOP`, `Ctrl+Z`) albo przez debugger
`Z`	zombie — proces zakończony, którego rodzic jeszcze nie odczytał kodu wyjścia
`I`	idle — wątek jądra w bezczynności (Linux ≥ 2.6.33)

Modyfikatory doklejane po literze stanu:

Modyfikator	Znaczenie
`<`	wysoki priorytet (ujemny nice)
`N`	niski priorytet (dodatni nice)
`L`	ma strony zablokowane w pamięci (mlock)
`s`	lider sesji
`l`	wielowątkowy (multi-threaded)
`+`	należy do grupy procesów na pierwszym planie terminala

`Ss+` czytasz więc: proces śpi (`S`), jest liderem sesji (`s`), i jest procesem pierwszoplanowym w terminalu (`+`) — dokładnie tak wygląda powłoka logowania, na której siedzisz.

Krok po kroku

1. Trzy niezgodne składnie w jednym poleceniu.

`ps` obsługuje naraz trzy tradycje opcji, opisane wprost w jego man page:

```
ps aux      # BSD — bez myślnika, litery sklejone
ps -ef      # UNIX/POSIX — z myślnikiem
ps --sort=-pcpu   # GNU długa opcja — dwa myślniki
```

To nie jest kosmetyka. `aux` w składni BSD znaczy: `a` — procesy wszystkich użytkowników z terminalem, `u` — format ze szczegółami (user-oriented), `x` — dołącz też procesy bez przypisanego terminala (demony). `-ef` w składni UNIX znaczy: `-e` — wszystkie procesy, `-f` — pełny format (full), z widocznym PPID. Dają podobny efekt końcowy (widzisz wszystko), ale różnymi kolumnami.

2. Dlaczego `ps aux` to nie to samo co `ps -aux`.

Dopisanie myślnika przed `aux` przełącza interpretację liter ze składni BSD na składnię UNIX. W UNIX-owej składni opcja `u` **oczekuje argumentu** — listy użytkowników. `-aux` bywa więc odczytane jako „`-a`, `-u` z argumentem `x`" — czyli próba pokazania procesów użytkownika o nazwie `x`. Man page `ps` opisuje to wprost jako pułapkę międzyepokową odziedziczoną po łączeniu tradycji BSD i POSIX. Współczesny `procps-ng` (RHEL 9/10, Debian, SLES) jest tu tolerancyjny i zwykle mimo wszystko pokaże to, co chciałeś — ale nie polegaj na tej tolerancji w skryptach. Pisz `ps aux` albo `ps -ef`, nigdy hybrydy.

3. Własny format zamiast pamiętania, co znaczy `%MEM`.

```
ps -o pid,ppid,user,stat,etime,rss,pcpu,args,wchan:20,nlwp --sort=-pcpu | head -10
```

```
    PID   PPID USER     STAT     ELAPSED   RSS %CPU COMMAND
WCHAN                NLWP
   4821      1 nginx    S           3-02:14:07  8420  0.3 nginx: worker process
ep_poll                 1
   1203      1 postgres Ss          3-02:15:01 94212  1.8 postgres: writer process   ...
```

`-o` daje pełną kontrolę: `etime` to czas życia procesu (elapsed), `wchan` mówi, na jaką funkcję jądra proces czeka (przydatne przy diagnozowaniu D-state), `nlwp` to liczba wątków. Kolumny w `-o` podajesz przecinkami, bez spacji — spacja rozdzieliłaby argument na dwa.

4. Drzewo zamiast płaskiej listy.

```
ps -ef --forest
```

```
root        1234       1  0 09:12 ?        00:00:00 /usr/sbin/httpd
apache      1240    1234  0 09:12 ?        00:00:02  \_ /usr/sbin/httpd
apache      1241    1234  0 09:12 ?        00:00:01  \_ /usr/sbin/httpd
```

Widzisz od razu, że dwa procesy `apache` są dziećmi procesu głównego `httpd` — kreski `\_` rysują hierarchię. Równoważne: `ps axjf` (składnia BSD z kolumnami zadania: PPID, PID, PGID, SID). Jeśli wolisz osobne narzędzie: `pstree -p adm_user` rysuje to samo bez kolumn liczbowych, samym drzewem nazw — na Debianie/Ubuntu i SLES pakiet nazywa się `psmisc`, tak samo jak na RHEL/Rocky.

5. Wątki widoczne osobno.

```
ps -eLf | awk 'NR==1 || $4==1203'
```

```
UID          PID    PPID     LWP  C NLWP STIME TTY          TIME CMD
postgres    1203       1    1203  0   12 09:10 ?        00:00:01 postgres: writer
postgres    1203       1    1207  0   12 09:10 ?        00:00:03 postgres: writer
```

`-L` dokłada kolumnę `LWP` (light-weight process, czyli wątek) i pokazuje każdy wątek jako osobny wiersz. Bez `-L` widziałbyś tylko jeden wiersz na proces `1203`, niezależnie od tego, ile ma wątków — a dla JVM czy bazy danych to bywa różnica między jednym wierszem a kilkuset.

6. Szukanie procesu bez łapania samego siebie.

```
pgrep -a nginx
```

```
4821 nginx: worker process
4822 nginx: worker process
```

`pgrep -a` (`-a` dopisuje pełną linię poleceń) filtruje procesy jądrowo, bez przechodzenia przez `grep` w ogóle — więc problem złapania samego siebie nie istnieje. Jeśli wolisz zostać przy `ps | grep`, mechanikę tricku z nawiasem kwadratowym i pełne omówienie wyrażeń regularnych znajdziesz na stronie *grep i rodzina: grep, egrep, fgrep, rekurencja*.

7. Ręczne zejście do `/proc`, gdy `ps` czegoś nie pokazuje.

```
PID=$(pgrep -o sshd)
cat /proc/$PID/cmdline | tr '\0' ' '; echo
readlink /proc/$PID/exe
ls -l /proc/$PID/fd | head
```

```
/usr/sbin/sshd -D [listener] 0 of 10-100 startups
/usr/sbin/sshd
lrwx------ 1 root root 64 wrz 17 09:10 0 -> /dev/null
lrwx------ 1 root root 64 wrz 17 09:10 3 -> socket:[28841]
```

`readlink /proc/PID/exe` pokazuje, jaki plik binarny proces faktycznie uruchomił — nawet jeśli plik na dysku został od tego czasu podmieniony albo usunięty (dowiązanie będzie miało dopisek `(deleted)`, klasyczny sygnał, że trzeba zrestartować usługę po aktualizacji pakietu).

Częste błędy

Grep łapie samego siebie.

```
$ ps aux | grep nginx
adm_user   8842  0.0  0.0   7908  3392 pts/0    S+   14:02   0:00 grep --color=auto nginx
```

Zero procesów `nginx`, za to widać `grep`, który go szukał — bo linia poleceń samego `grep nginx` też zawiera napis `nginx`. Dwa poprawne obejścia:

```
ps aux | grep '[n]ginx'     # nawias łamie dopasowanie do samego siebie
pgrep -a nginx              # bez grepa w ogóle — lepsze rozwiązanie
```

Trik z nawiasem działa, bo `grep` szuka teraz wyrażenia `[n]ginx`, a linia procesu `grep` zawiera dosłowny tekst `[n]ginx` (z nawiasami), nie `nginx` — więc się nie dopasowuje do samej siebie.

Odczytanie VSZ jako realnego zużycia pamięci.

```
$ ps aux | awk '$2==1203{print $5}'
1848320
```

To 1,8 GB w kolumnie VSZ, po czym pada raport „proces zżera prawie 2 GB RAM-u". Proces może mieć taki VSZ i faktycznie trzymać w RAM-ie 90 MB (kolumna RSS). VSZ liczy całą zaadresowaną przestrzeń, łącznie z niedotkniętymi mapowaniami. Do realnego zużycia patrz na RSS, a jeszcze lepiej na PSS z `/proc/PID/smaps_rollup`.

Sumowanie RSS jako całkowitego zużycia RAM-u.

```
ps -eo rss | awk '{sum+=$1} END {print sum/1024 " MB"}'
```

Wynik potrafi przekroczyć fizyczny RAM maszyny, bo strony bibliotek współdzielonych (np. `libc`, `libssl`) liczą się osobno w RSS każdego procesu, który ich używa. Do prawdziwego obrazu pamięci systemowej służy `free -h`, nie suma po `ps`.

`ps -aux` traktowane jako literalnie to samo co `ps aux`.

W skrypcie automatyzującym ktoś wpisuje `ps -aux` z przyzwyczajenia. Zadziała bez ostrzeżenia dzięki tolerancji `procps-ng`, ale to zachowanie nie jest gwarantowane przez POSIX i bywa inne na uboższych implementacjach `ps` (np. w obrazach kontenerowych opartych na BusyBox). Pisz `ps aux` albo `ps -ef` — nie hybrydę.

Odpalenie własnej pętli monitorującej zamiast gotowego narzędzia.

```
while true; do ps aux | grep nazwa_procesu; sleep 2; done
```

Działa, ale uruchamia `ps` i `grep` jako nowe procesy co dwie sekundy, w nieskończonej pętli bez warunku wyjścia — zabijesz to tylko `Ctrl+C`. To zadanie ma gotowe narzędzia: `watch -n 2 'pgrep -a nazwa_procesu'` robi to samo, czyściej, z timestampem i możliwością wyjścia klawiszem `q` po dodaniu odpowiedniej flagi w nowszych wersjach `watch`. Do stałego monitorowania z odświeżaniem żywym — `top` albo `htop`, temat następnej strony.

□□ Ćwiczenia

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

Ćwiczenie 1 — namierzanie własnego procesu trzema sposobami

```
cd /tmp

# 1. Odpal proces w tle, który pożyje 5 minut
sleep 300 &
PID=$!
echo "PID nowego procesu: $PID"

# 2. Znajdź go trzema różnymi sposobami
ps aux | grep "[s]leep 300"
pgrep -a sleep
ps -p $PID -o pid,ppid,stat,etime,args

# 3. Zajrzyj do /proc ręcznie
cat /proc/$PID/cmdline | tr '\0' ' '; echo
readlink /proc/$PID/exe
cat /proc/$PID/status | grep -E '^(State|PPid|VmRSS|VmSize)'

# 4. Sprzątnij
kill $PID
```

Ile z trzech sposobów w kroku 2 zwróci ten sam PID, i co dokładnie pokazuje `VmSize` w kroku 3 w porównaniu z tym, co `ps` nazywa `VSZ`?

Odpowiedź

Krok 2 zwraca ten sam PID trzema drogami. W kroku 3 `VmSize` w `/proc/PID/status` to dokładnie ta liczba, którą `ps` pokazuje jako `VSZ` (w kilobajtach), a `VmRSS` to `RSS`. `State` pokaże `S (sleeping)` — `sleep` śpi, czekając na upłynięcie czasu, więc jest w przerywalnym uśpieniu, nie w `D`.

Po `kill $PID` proces znika z listy natychmiast — `sleep` reaguje na `SIGTERM` domyślnie wysyłany przez `kill`.

Pytania kontrolne

1. Proces ma VSZ = 2 GB i RSS = 40 MB. Czy to oznacza awarię albo wyciek pamięci?

Odpowiedź

Niekoniecznie. VSZ to cała zaadresowana przestrzeń — może obejmować duże mapowanie pliku (`mmap`) albo zarezerwowaną, ale nietkniętą pamięć. RSS mówi, ile faktycznie leży w RAM-ie. Różnica rzędu dziesiątek razy jest normalna dla procesów, które mapują duże pliki lub biblioteki, ale dotykają tylko ich skrawka. Sygnałem alarmowym byłby rosnący w czasie RSS, nie sama różnica VSZ/RSS w jednym zdjęciu.

2. Dlaczego `ps aux | grep apache` czasem pokazuje proces, którego w systemie nie ma?

Odpowiedź

Bo wynik zawiera też sam proces `grep apache` — jego linia poleceń zawiera szukany napis. To nie jest proces `apache`, tylko `grep` przeszukujący dane. Rozwiązanie: `grep '[a]pache'` albo `pgrep -a apache`.

3. Masz dwie komendy: `ps aux --sort=-vsz` i `ps aux --sort=-rss`. Czy zwrócą ten sam proces na szczycie listy?

Odpowiedź

Niekoniecznie. Proces z największym VSZ (np. taki, który zamapował ogromny plik) może mieć niski RSS, jeśli większości tego mapowania nigdy nie dotknął. Proces z największym RSS to ten, który faktycznie zajmuje najwięcej fizycznego RAM-u — i to jest pytanie, które zwykle naprawdę Cię interesuje przy szukaniu winowajcy zapchanej pamięci.

4. Widzisz w kolumnie STAT wartość `Ss+`. Rozbij to na czynniki pierwsze.

Odpowiedź

`S` — proces w przerywalnym uśpieniu (czeka na zdarzenie, reaguje na sygnały). `s` — jest liderem sesji. `+` — należy do grupy procesów na pierwszym planie terminala. Razem: to typowy obraz powłoki interaktywnej, na której właśnie pracujesz — śpi, czekając na Twoje wejście, jest głową swojej sesji terminala i siedzi na pierwszym planie.

Powiązane: Monitorowanie na żywo: top, htop, btop · Stany procesów: zombie, sieroty, D-state · grep i rodzina: grep, egrep, fgrep, rekurencja · Sygnały: kill, pkill, killall, trap

Źródła: `man 1 ps` (sekcja NOTES — `ps -aux` vs `ps aux`) · `man 5 proc` · `man 1 pgrep` · `man 1 pstree`

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