Pamięć naprawdę: free i /proc/meminfo

Poziom: średnio **Dotyczy: wszystkie dystrybucje** **Czas: ~18 min**

`free` *pokazuje bilans pamięci RAM i swap w jednej chwili. Większość alarmów „serwer się dławi z pamięci" bierze się z odczytania niewłaściwej kolumny — ta strona uczy, którą kolumnę naprawdę czytać, i co zrobić, gdy rzeczywiście trzeba.*

Skróty użyte na tej stronie (6)

Skrót	Co znaczy
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
GiB	gibibajt: 1024 MiB, czyli około 1,07 GB; tej jednostki używa dokumentacja MailCow przy wymaganiach pamięci
OOM	Out Of Memory: brak pamięci; OOM killer to mechanizm jądra zabijający proces, gdy jej zabraknie
PID	Process ID: numer procesu w systemie
RAM	Random Access Memory: pamięć operacyjna
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

Fiszka

TYLKO ODCZYT

Wypisanie stanu pamięci bez żadnych flag

```
free
```

Bez flag wynik jest w kilobajtach — najmniej czytelny, ale to jest format bazowy, na którym rozumowuje reszta narzędzia.

Przykład:

```
               total        used        free      shared  buff/cache   available
Mem:        16308000     8598000     2150000      512000     5560000     7412000
Swap:        4194300      512000     3682300
```

free = pokaż ile pamięci RAM i przestrzeni wymiany jest wolne, a ile zajęte

TYLKO ODCZYT

Wypisanie stanu pamięci w formacie czytelnym dla człowieka: -h

```
free -h
```

`-h` (human-readable) przelicza bajty na jednostki dopasowane do wielkości liczby — B, KiB, MiB, GiB. To jest wariant, po który sięgamy w 95% przypadków.

Przykład:

```
               total        used        free      shared  buff/cache   available
Mem:            15Gi       8.2Gi       2.1Gi       512Mi       5.3Gi       7.4Gi
Swap:          4.0Gi       512Mi       3.5Gi
```

Zanim ta tabela zacznie coś znaczyć, każda kolumna osobno:

Kolumna	Co znaczy
`total`	cała pamięć fizyczna, jaką widzi jądro
`used`	pamięć aktualnie przypisana procesom (`total` minus reszta kolumn, w uproszczeniu)
`free`	pamięć zupełnie niedotknięta — nikt jej nie używa, nawet cache
`shared`	pamięć współdzielona między procesami (m.in. `tmpfs`, segmenty `/dev/shm`)
`buff/cache`	bufory jądra i pamięć podręczna stron — patrz ramka niżej
`available`	szacunek jądra: ile pamięci realnie dostanie nowa aplikacja bez sięgania po swap

> **Pojęcie: pamięć podręczna stron (page cache).** Kiedy proces czyta plik z dysku, jądro trzyma przeczytane bloki w pamięci RAM jeszcze długo po tym, jak proces

skończył czytanie — na wypadek, gdyby ktoś zapytał o te same dane ponownie. To jest page cache, widoczny w kolumnie `buff/cache`. Nie jest to pamięć „zajęta" w sensie, w jakim zajęta jest pamięć procesu — jądro oddaje ją natychmiast, gdy jakaś aplikacja faktycznie jej potrzebuje. Serwer, który od tygodnia nie był restartowany, ma zwykle prawie całą wolną pamięć zamienioną w cache, bo jądro uznaje: skoro nikt o nią nie prosi, lepiej trzymać w niej coś użytecznego niż nic.

Pojęcie: pamięć dostępna kontra wolna. `free` (wolna) to pamięć, której dosłownie nikt nie dotyka — ani proces, ani cache. `available` (dostępna) to znacznie użyteczniejsza liczba: szacunek jądra, ile pamięci dostanie nowo uruchomiona aplikacja, licząc również tę część `buff/cache`, którą da się natychmiast odzyskać. Kolumna `available` pojawiła się w `free` dopiero z kernelem 3.14 (2014) i `procps` w wersji obsługującej `/proc/meminfo` pole `MemAvailable` — na bardzo starych systemach jej brak, wtedy trzeba liczyć ręcznie z `free` i `buff/cache`. Na każdym systemie młodszym niż kilka lat to jest jedyna kolumna, którą warto śledzić przy pytaniu „czy zabraknie mi pamięci".

GiB = gibibajt: 1024 MiB, czyli około 1,07 GB; tej jednostki używa dokumentacja MailCow przy wymaganiach pamięci · **RAM** = Random Access Memory: pamięć operacyjna · **free** = pokaż ile pamięci RAM i przestrzeni wymiany jest wolne, a ile zajęte

TYLKO ODCZYT

Wypisanie stanu pamięci w megabajtach: -m

```
free -m
```

Wymuszony format w MB, niezależnie od wielkości liczby. Przydatne w skryptach, gdzie chcemy stałej jednostki do porównań liczbowych, a `-h` dawałoby raz `Mi`, raz `Gi` w zależności od wartości.

Przykład:

```
               total        used        free      shared  buff/cache   available
Mem:           15925        8396        2099         500        5429        7238
Swap:           4095         500        3596
```

free = pokaż ile pamięci RAM i przestrzeni wymiany jest wolne, a ile zajęte

TYLKO ODCZYT

Wypisanie stanu pamięci w gigabajtach: -g

```
free -g
```

To samo co `-m`, ale w GB, z zaokrągleniem w dół do pełnych jednostek — na maszynie z 16 GB RAM-u i drobnym zużyciem `used` może pokazać `0`, bo ułamki giga znikają. Sensowne tylko na naprawdę dużych serwerach, gdzie różnica kilkuset MB i tak ginie w szumie.

Przykład:

```
               total        used        free      shared  buff/cache   available
Mem:              15           8           2           0           5           7
Swap:              3           0           3
```

GB = Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej · **free** = pokaż ile pamięci RAM i przestrzeni wymiany jest wolne, a ile zajęte

TYLKO ODCZYT

Rozbicie buforów i cache na dwie kolumny: -w

```
free -w
```

Domyślnie `buff/cache` to jedna zlepiona kolumna. `-w` (wide) rozdziela ją na `buffers` (metadane systemu plików, małe) i `cache` (właściwy page cache plików, zwykle znacznie większy). Sięgamy po to, gdy podejrzewamy konkretnie system plików, a nie cache treści plików.

Przykład:

```
               total        used        free      shared     buffers       cache
available
Mem:            15Gi       8.2Gi       2.1Gi       512Mi       180Mi       5.1Gi
7.4Gi
Swap:          4.0Gi       512Mi       3.5Gi
```

free = pokaż ile pamięci RAM i przestrzeni wymiany jest wolne, a ile zajęte

TYLKO ODCZYT

Odświeżanie wyniku co interwał: -s

```
free -s 2
```

Powtarza pomiar co 2 sekundy w nieskończoność — przerywamy `Ctrl+C`. Dobre do obserwowania trendu w czasie rzeczywistym, np. podczas wgrywania dużego importu danych.

Przykład:

```
               total        used        free      shared  buff/cache   available
Mem:            15Gi       8.2Gi       2.1Gi       512Mi       5.3Gi       7.4Gi
Swap:          4.0Gi       512Mi       3.5Gi

               total        used        free      shared  buff/cache   available
Mem:            15Gi       8.4Gi       1.9Gi       512Mi       5.3Gi       7.2Gi
Swap:          4.0Gi       512Mi       3.5Gi
```

Dodanie `-c 10` (`free -h -s 2 -c 10`) ogranicza liczbę powtórzeń do dziesięciu i kończy działanie samo, bez `Ctrl+C` — wygodniejsze w logach ze skryptu diagnostycznego niż nieskończona pętla.

free = pokaż ile pamięci RAM i przestrzeni wymiany jest wolne, a ile zajęte

TYLKO ODCZYT

Odświeżanie ekranu z podświetleniem zmian: watch -n 1 'free -h'

```
watch -n 1 'free -h'
```

`watch` (osobny pakiet, `procps-ng` na RHEL/Rocky, ten sam pakiet co `free`) odświeża cały ekran co sekundę i podświetla różnice względem poprzedniego odczytu — czytelniejsze niż przewijający się strumień z `free -s`, ale zajmuje cały terminal.

RHEL = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **watch** = uruchamiaj wskazane polecenie co kilka sekund i pokazuj wynik na żywo

TYLKO ODCZYT

Wypisanie surowych danych źródłowych polecenia free: cat /proc/meminfo

```
cat /proc/meminfo
```

`free` niczego nie mierzy samodzielnie — po prostu odczytuje ten plik i przelicza wybrane pola. Warto znać go bezpośrednio, bo ma dziesiątki pól, których `free` w ogóle nie pokazuje.

Przykład:

```
MemTotal:       16308000 kB
MemFree:         2150000 kB
MemAvailable:    7412000 kB
Buffers:          184000 kB
Cached:          5210000 kB
SwapCached:            0 kB
Active:          6120000 kB
Inactive:        4890000 kB
Dirty:              8400 kB
Writeback:             0 kB
SwapTotal:       4194300 kB
SwapFree:        3682300 kB
Slab:             612000 kB
```

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Wypisanie wybranych pól z /proc/meminfo: grep -E

```
grep -E 'MemTotal|MemAvailable|Dirty|Writeback' /proc/meminfo
```

Cztery pola, które w praktyce sprawdza się najczęściej. `Dirty` to strony pamięci zmodyfikowane, ale jeszcze niezapisane na dysk — jądro odkłada zapis, żeby zebrać więcej zmian naraz i zapisać je efektywniej. `Writeback` to strony aktualnie w trakcie zapisu na dysk. Rosnący, niemalejący `Dirty` przy niskim `Writeback` bywa sygnałem, że dysk nie nadąża odbierać danych, które jądro chce zrzucić.

Przykład:

```
MemTotal:       16308000 kB
MemAvailable:    7412000 kB
Dirty:              8400 kB
Writeback:             0 kB
```

grep = global regular expression print: szukaj wzorca w tekście

TYLKO ODCZYT

Wyszukanie procesów zajmujących najwięcej pamięci: ps aux --sort=-%mem

```
ps aux --sort=-%mem | head -6
```

Gdy `available` jest niepokojąco niskie, następny krok to znalezienie winowajcy. Pełny opis kolumn i pułapek tego polecenia (m.in. sumowanie `RSS`, które zawyża realne zużycie) jest na stronie o `ps`.

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
java      2044  4.2  4.1 3102400 662400 ?      Sl   08:05  22:40 java -jar app.jar
root         1  0.0  0.1  171004  11840 ?      Ss   wrz16   0:03
/usr/lib/systemd/systemd
```

RSS = Resident Set Size: ile pamięci RAM proces faktycznie zajmuje w tej chwili, w odróżnieniu od VSZ · **ps** = process status: pokaż procesy

TYLKO ODCZYT

Sprawdzenie realnego zużycia pamięci z rozbiciem współdzielonej: smem

```
smem -t -k
```

`ps` i `top` liczą `RSS`, w którym biblioteki współdzielone (np. `libc`) doliczają się w pełnej wysokości do każdego procesu, który ich używa. `smem` (osobny pakiet, doinstalowywany ręcznie — nie ma go domyślnie na żadnej z głównych dystrybucji) liczy `PSS` (Proportional Set Size), dzieląc współdzieloną pamięć proporcjonalnie między procesy, więc suma wszystkich wierszy realnie zbliża się do zużycia systemowego.

Przykład:

```
  PID User     Command                         Swap      USS      PSS      RSS
 1203 postgres postgres: writer process           0   812000   860400   943212
 2044 java     java -jar app.jar                  0   598000   615200   662400
```

```
--------------------------------------------------------------------------------
                                                      0  1410000  1475600  1605612
```

RSS = Resident Set Size: ile pamięci RAM proces faktycznie zajmuje w tej chwili, w odróżnieniu od VSZ

TYLKO ODCZYT

Sprawdzenie, czy OOM killer już działał: dmesg -T

```
dmesg -T | grep -i 'out of memory'
```

`-T` tłumaczy znaczniki czasu jądra na czytelną datę zamiast sekund od startu systemu. Jeśli coś się pojawi, system w przeszłości zabrakło pamięci na tyle mocno, że jądro musiało ubić proces — pełny rozbiór komunikatu jest w sekcji *Jak to działa*.

Przykład:

```
[Tue Sep 16 03:12:41 2026] Out of memory: Killed process 20441 (java) total-
vm:3102400kB, anon-rss:2841200kB, file-rss:0kB, shmem-rss:0kB, UID:993 pgtables:5680kB
oom_score_adj:0
```

OOM = Out Of Memory: brak pamięci; OOM killer to mechanizm jądra zabijający proces, gdy jej zabraknie · **dmesg** = pokaż komunikaty jądra, przydatne przy diagnozowaniu sprzętu i dysków

TYLKO ODCZYT

Sprawdzenie działania OOM killera z dziennika systemd: journalctl -k

```
journalctl -k | grep -i oom
```

Na systemach z `systemd-journald` to jest odpowiednik `dmesg`, z dodatkowym atutem: przetrwa restart, jeśli journal jest skonfigurowany trwale (`Storage=persistent` w `/etc/systemd/journald.conf`), podczas gdy bufor `dmesg` czyści się przy każdym restarcie.

Przykład:

```
wrz 16 03:12:41 srv01 kernel: java invoked oom-killer:
gfp_mask=0x140cca(GFP_HIGHUSER_MOVABLE), order=0, oom_score_adj=0
wrz 16 03:12:41 srv01 kernel: Out of memory: Killed process 20441 (java) total-
vm:3102400kB, anon-rss:2841200kB, file-rss:0kB, shmem-rss:0kB
```

OOM = Out Of Memory: brak pamięci; OOM killer to mechanizm jądra zabijający proces, gdy jej zabraknie · **journalctl** = journal control: czytanie dziennika systemd

TYLKO ODCZYT

Sprawdzenie pamięci jednego konkretnego procesu: cat /proc/NumerPID/status

```
cat /proc/NumerPID/status | grep Vm
```

> (gdzie `NumerPID` to identyfikator procesu z wyniku `ps aux` albo `top`)

Wszystkie liczniki pamięci procesu naraz, prosto z jądra — bez przechodzenia przez `ps`. `VmRSS` to to samo, co `ps` pokazuje jako `RSS`; `VmSwap` mówi, ile z tego procesu zostało wypchnięte do swapu.

Przykład:

```
VmPeak:  3102400 kB
VmSize:  3102400 kB
VmRSS:    662400 kB
VmData:   580000 kB
VmSwap:        0 kB
```

RSS = Resident Set Size: ile pamięci RAM proces faktycznie zajmuje w tej chwili, w odróżnieniu od VSZ · **cat** = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Sprawdzenie pamięci per cgroup zamiast per proces: systemd-cgtop

```
systemd-cgtop -m
```

Grupuje zużycie pamięci po jednostkach systemd (usługach, slice'ach), nie po pojedynczych PID-ach. Przydatne, gdy usługa uruchamia wiele procesów potomnych i interesuje nas suma całej jednostki, nie rozbicie na pojedyncze `postgres: writer`, `postgres: checkpointer` itd.

Przykład:

```
Control Group                    Tasks   %CPU   Memory  Input/s Output/s
/system.slice/postgresql.service    14    3.2    1.4G        -        -
/system.slice/nginx.service          5    0.1   48.2M        -        -
/user.slice                         22    0.4  210.5M        -        -
```

□□ TYLKO ODCZYT

Wszystkie flagi

Flaga	Co robi
`-h`	jednostki dopasowane do wielkości (KiB/MiB/GiB)
`-m`	wymuszony wynik w MB
`-g`	wymuszony wynik w GB
`-w`	rozbija `buff/cache` na osobne `buffers` i `cache`
`-s N`	odświeża wynik co N sekund, w nieskończoność
`-c N`	z `-s`: kończy po N powtórzeniach

Uważaj

- Niski `free` to normalny stan zdrowego, długo działającego systemu — jądro celowo zamienia niewykorzystaną pamięć w cache.
- `free -s` bez `-c` działa bez końca; w skrypcie zawsze dopisz `-c`, żeby nie zawiesić joba.
- Suma `RSS` z `ps` po wielu procesach zawyża realne zużycie pamięci — do sumowania służy `smem` (PSS), nie `ps`.

□□ **RSS** = Resident Set Size: ile pamięci RAM proces faktycznie zajmuje w tej chwili, w odróżnieniu od VSZ

Z życia wzięte

□□ Z ŻYCIA WZIĘTE

Środa, 11:00. Aplikacja Javy zespołu Oli na app01 zaczyna być zabijana przez system w środku dnia, bez żadnego komunikatu w swoich własnych logach. Tomek sprawdza free i

> widzi, że mimo dużej ilości pamięci podręcznej system rzeczywiście wyczerpał pamięć dostępną dla procesów, a OOM killer po prostu wybrał największego konsumenta.

Po co to istnieje

> **Pojęcie: OOM killer.** Gdy jądro Linuksa nie potrafi już przydzielić pamięci — ani z wolnej puli, ani przez odzyskanie cache, ani przez wypchnięcie stron do swapu — uruchamia mechanizm awaryjny zwany OOM killerem (Out-Of-Memory killer). Zamiast pozwolić całemu systemowi się zawiesić, wybiera jeden proces według wyliczonego `oom_score` (im więcej pamięci proces zajmuje i im mniej jest „ważny" wg heurystyk jądra, tym wyższy wynik) i zabija go sygnałem, którego nie da się zignorować. To jest ostatnia linia obrony, nie codzienny mechanizm — jej uruchomienie się zawsze znaczy, że system realnie wyczerpał pamięć, a nie że coś tylko wygląda niepokojąco w `free`.

Administrator sprawdza pamięć z jednego konkretnego powodu praktycznego: chce wiedzieć, czy system ma zapas, zanim uruchomi coś ciężkiego (import bazy, kompilację, backup), i czy niedawny restart usługi nie był efektem OOM killera, a nie awarii aplikacji. `free` odpowiada na pierwsze pytanie w pięć sekund, `dmesg` / `journalctl` na drugie.

Analogia

Wyobraź sobie **bibliotekę z czytelnią**, w której bibliotekarz zamiast odkładać zwrócone książki od razu na półkę, trzyma je chwilę na wózku obok lady.

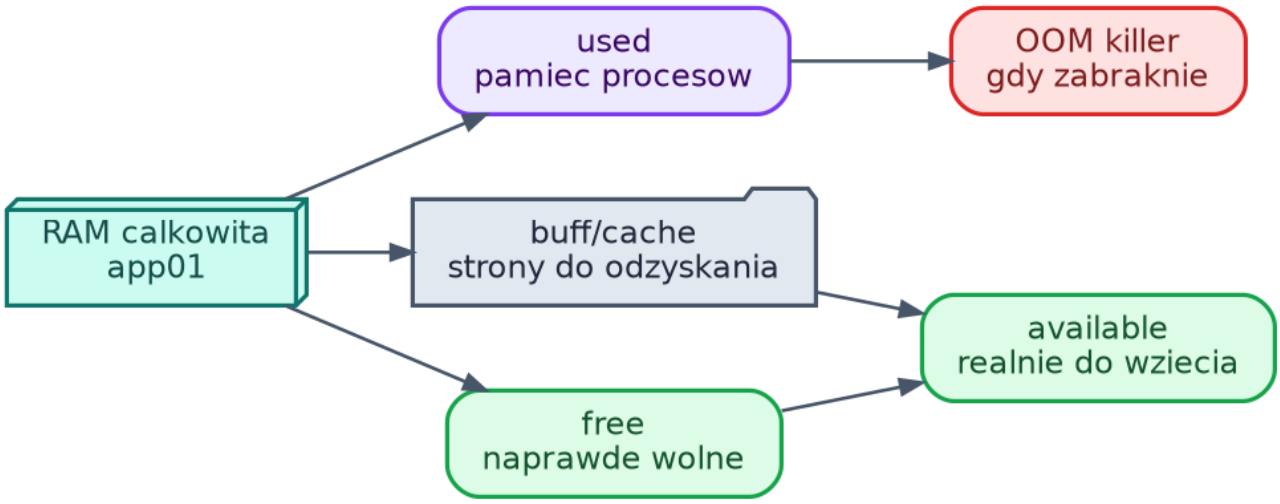
Kolejny czytelnik, który poprosi o tę samą książkę, dostanie ją natychmiast z wózka, zamiast czekać, aż bibliotekarz odszuka ją na regale. Wózek zawsze jest pełny książek — to nie jest bałagan, to zamierzona strategia: skoro miejsce na wózku i tak stało puste, lepiej trzymać na nim coś przydatnego. Gdy jednak przyjdzie nowy czytelnik z zupełnie inną prośbą, a wózek jest zapełniony, bibliotekarz w sekundę zrzuca z niego dowolną książkę na regał i zwalnia miejsce — nikt na to nie czeka.

`free` (wolne miejsce na blacie, dosłownie nic) to nie to samo co `available` (to, co bibliotekarz odda Ci natychmiast, licząc też wózek). Kolumna `used` to książki faktycznie w rękach czytelników w czytelni — te bibliotekarz odda dopiero, gdy czytelnik skończy. A OOM killer to sytuacja, w której przychodzi więcej czytelników, niż biblioteka ma w ogóle miejsc siedzących, i ochrona musi kogoś fizycznie wyprosić, żeby reszta mogła w ogóle wejść.

Jak to działa

Rozwiń: jak to działa pod spodem

Jądro liczy `MemAvailable` (a `free` pokazuje ją jako `available`) w przybliżeniu jako sumę:



Pamięć RAM dzieli się na zajętą przez procesy, cache do odzyskania i naprawdę wolną, zanim wkroczy OOM killer. (kliknięcie otwiera pełny rozmiar)

- pamięci zupełnie wolnej (`MemFree`),
- części page cache, którą da się odzyskać bez zapisu na dysk (strony, które nie są „brudne" — patrz `Dirty` w `/proc/meminfo`),
- części pamięci reclaimable ze struktur jądra (np. część `Slab`).

To jest przybliżenie, nie gwarancja co do bajta — ale znacznie bliższe prawdy niż surowe `free`. Zanim to pole istniało (kernele starsze niż 3.14, czyli praktycznie żaden system spotykany dziś w produkcji), administratorzy liczyli to ręcznie jako `free + buffers + cache`, co dawało zawyżony wynik, bo nie każdy fragment cache da się odzyskać natychmiast bez oczekiwania na zapis brudnych stron.

Ścieżka do OOM killera wygląda tak: proces prosi jądro o więcej pamięci (`brk()`, `mmap()`, czasem po prostu przez zapis do już zaalokowanej strony, co wywołuje page fault). Jądro próbuje po kolei: przydzielić z wolnej puli, odzyskać część page cache, wypchnąć nieaktywne strony do swapu (jeśli jest miejsce — patrz strona o SWAP). Jeśli żadna z tych dróg nie wystarcza, wywoływana jest funkcja `out_of_memory()`, która przechodzi po wszystkich

procesach, liczy `oom_score` dla każdego (uwzględniając m.in. zużycie pamięci, czas działania, `oom_score_adj` ustawiony ręcznie) i wysyła `SIGKILL` do procesu z najwyższym wynikiem.

Rozbiór realnego komunikatu z `dmesg`:

```
[Tue Sep 16 03:12:41 2026] Out of memory: Killed process 20441 (java) total-vm:3102400kB,
anon-rss:2841200kB, file-rss:0kB, shmem-rss:0kB, UID:993 pgtables:5680kB oom_score_adj:0
```

- `Killed process 20441 (java)` — PID i nazwa procesu, który padł ofiarą. To jest ofiara, niekoniecznie przyczyna: proces, który *spowodował* wyczerpanie pamięci, mógł już wcześniej zwolnić ją i zniknąć z listy, a padł ten, który akurat miał najwyższy `oom_score` w chwili decyzji.
- `total-vm:3102400kB` — to samo co `VSZ` w `ps`, cała zaadresowana przestrzeń.
- `anon-rss:2841200kB` — pamięć anonimowa (sterta, stos) faktycznie w RAM-ie, bez mapowań plikowych. To zwykle najlepszy wskaźnik „ile ten proces realnie pochłonął".
- `oom_score_adj:0` — wartość, o którą administrator mógł ręcznie podbić lub obniżyć wynik tego procesu (zakres od `-1000`, czyli „nigdy nie zabijaj", do `1000`); `0` znaczy brak ręcznej ingerencji.

Linia wyżej w logu (`java invoked oom-killer: ...`) mówi, **który proces poprosił o pamięć w chwili, gdy zabrakło jej dla wszystkich** — to bywa ten sam proces, który potem zginął, ale nie musi.

Krok po kroku

1. Szybki test zdrowia — jedno polecenie.

```
free -h
```

```
               total        used        free      shared  buff/cache   available
Mem:            15Gi       8.2Gi       2.1Gi       512Mi       5.3Gi       7.4Gi
Swap:          4.0Gi       512Mi       3.5Gi
```

Patrzysz wyłącznie na `available` (7.4Gi) w relacji do `total` (15Gi). Tu jest prawie połowa pamięci realnie do dyspozycji — system zdrowy, mimo że `free` (2.1Gi) samo w sobie wygląda skromnie.

2. Jeśli `available` jest niepokojąco niskie, zajrzyj głębiej.

```
grep -E 'MemTotal|MemAvailable|Dirty|Writeback' /proc/meminfo
```

```
MemTotal:       16308000 kB
MemAvailable:     420000 kB
Dirty:            180000 kB
Writeback:         95000 kB
```

`MemAvailable` poniżej 3% `MemTotal` to już poziom, na którym warto działać, zanim zadziała OOM killer. Wysoki `Writeback` przy tym sugeruje, że dysk nie nadąża odprowadzać brudnych stron — problem może być zarówno pamięciowy, jak i dyskowy naraz.

3. Znajdź, kto zabiera pamięć.

```
ps aux --sort=-%mem | head -6
```

Kolumna `%MEM` daje szybki ranking; przy podejrzeniu na biblioteki współdzielone lepszym źródłem jest `smem -t -k`.

4. Sprawdź, czy to wzrost, czy stały poziom.

```
watch -n 5 'cat /proc/2044/status | grep VmRSS'
```

```
VmRSS:□  662400 kB
```

Jeśli `VmRSS` rośnie w kolejnych odczytach bez spadków, masz do czynienia z wyciekiem pamięci (memory leak), nie z chwilowym szczytem obciążenia.

5. Sprawdź, czy OOM killer już wcześniej interweniował.

```
journalctl -k -b | grep -i oom
```

Puste — nikt nie był zabijany od ostatniego rozruchu. Niepuste — masz historyczny dowód i konkretny PID/nazwę procesu do dalszego dochodzenia.

Częste błędy

Panika na widok niskiej kolumny `free`.

```
Mem:           15Gi       2.0Gi        95Mi       200Mi        13Gi        12Gi
```

`free` (druga kolumna od końca licząc od `used`) to tylko 95 MiB, co wygląda dramatycznie. Ale `available` to 12Gi — system ma się świetnie, po prostu prawie cała niewykorzystana pamięć została zamieniona w page cache. Naprawa: patrz wyłącznie na `available`, ignoruj `free` jako liczbę do oceny stanu systemu.

Traktowanie `buff/cache` jako pamięci nie do odzyskania.

Administrator widzi 5.3 GiB w `buff/cache` i uruchamia ręczne czyszczenie cache (`echo 3 > /proc/sys/vm/drop_caches`), żeby „zwolnić pamięć" przed uruchomieniem zadania. Efekt: kolejne odczyty tych samych plików muszą wracać na dysk, aplikacje na chwilę zwalniają, a `available` i tak było już wcześniej wystarczające. Pełny rozbiór, dlaczego to prawie nigdy nie jest właściwa reakcja, jest na stronie o SWAP.

Mylenie `RSS` z realnym zużyciem RAM po zsumowaniu procesów.

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

```
23.8 GB
```

Wynik przekracza fizyczny RAM maszyny (16 GB), bo biblioteki współdzielone liczą się w pełnej wysokości w `RSS` każdego procesu, który ich używa. Naprawa: `smem -t -k`, które liczy `PSS` (proporcjonalny udział), albo po prostu `free -h`.

Odczytanie `dmesg` bez `-T` i pomylenie czasu zdarzenia.

```
dmesg | grep -i oom
```

```
[1502341.884921] Out of memory: Killed process 20441 (java) total-vm:3102400kB...
```

Liczba w nawiasie to sekundy od startu systemu, nie znacznik czasu. Bez przeliczenia trudno powiązać zdarzenie z konkretną godziną w innych logach. Naprawa: zawsze `dmesg -T`, albo — na systemach z trwałym journalem — `journalctl -k`, gdzie czas jest czytelny od razu.

Zaufanie do `MemAvailable` na bardzo starym jądrze.

Na systemie z kernelem starszym niż 3.14 (spotykane już tylko na zapomnianych maszynach) pole `MemAvailable` w `/proc/meminfo` po prostu nie istnieje, a `free` w takiej sytuacji albo nie pokazuje kolumny `available` wcale, albo pokazuje przybliżenie liczone samodzielnie przez `procps`. Warto to sprawdzić jednym poleceniem przed zaufaniem liczbie: `grep MemAvailable /proc/meminfo` — brak wyniku oznacza, że trzeba liczyć ręcznie z `free`, `buffers` i `cache`.

□□ Ćwiczenia

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

Ćwiczenie 1 — czytanie prawdziwego wyniku `free -h`

Masz taki wynik `free -h` z serwera bazodanowego:

```
               total        used        free      shared  buff/cache   available
Mem:            31Gi        24Gi       420Mi       180Mi       6.8Gi       6.1Gi
Swap:          8.0Gi       3.2Gi       4.8Gi
```

Czy ten system ma problem z pamięcią? Uzasadnij, którą kolumnę bierzesz pod uwagę.

Odpowiedź

Sytuacja jest już napięta, choć niekatastrofalna. `available` to 6.1 GiB na 31 GiB total — około 20%, więc jeszcze jest zapas, ale znacznie mniejszy niż w zdrowym przykładzie z tej strony (tam było ~47%). Do tego swap ma w użyciu 3.2 GiB, co samo w sobie nie jest dowodem awarii (patrz strona o SWAP), ale w połączeniu z niskim `available` sugeruje, że warto sprawdzić `vmstat 2` pod kątem aktywnego `si/so`, zanim system sięgnie po OOM killera.

Ćwiczenie 2 — symulacja i obserwacja zużycia pamięci procesu

```
cd /tmp
python3 -c "
import time
data = []
for i in range(20):
    data.append(bytearray(20 * 1024 * 1024))  # 20 MB na iterację
    time.sleep(1)
time.sleep(60)
" &
PID=$!
echo "PID: $PID"
```

W drugim terminalu obserwuj wzrost:

```
watch -n 2 "cat /proc/$PID/status | grep -E 'VmRSS|VmSize'"
```

Po zaobserwowaniu wzrostu zakończ proces: `kill $PID`. Co różni `VmSize` od `VmRSS` w kolejnych odczytach?

Odpowiedź

`VmSize` (odpowiednik `VSZ`) rośnie skokowo za każdym razem, gdy proces alokuje nowy bufor — to cała zarezerwowana przestrzeń adresowa. `VmRSS` rośnie wolniej i bardziej płynnie, bo pokazuje ile z tych zaalokowanych bajtów faktycznie zostało zapisanych i trafiło do fizycznego RAM-u (Python zapisuje zera do każdego bajtu `bytearray` przy tworzeniu, więc tu obie liczby rosną niemal razem — w realnych aplikacjach, które tylko rezerwują pamięć bez dotykania jej, różnica bywa dużo większa).

Pytania kontrolne

1. Dlaczego kolumna `free` w `free -h` może pokazywać kilkaset MB na serwerze, który działa bez zarzutu od miesięcy?

Odpowiedź

Bo jądro celowo zamienia niewykorzystaną pamięć w page cache — przechowuje w niej ostatnio czytane pliki, żeby przyspieszyć kolejne odczyty. Ta pamięć jest widoczna w `buff/cache`, nie w `free`, ale jest odzyskiwalna natychmiast, gdy aplikacja jej zażąda. Do oceny zdrowia patrzy się na `available`, nie na `free`.

2. Co dokładnie zabija OOM killer i według jakiego kryterium wybiera ofiarę?

Odpowiedź

Zabija jeden proces sygnałem `SIGKILL`, wybrany na podstawie wyliczonego `oom_score` — im więcej pamięci proces zajmuje i im mniej jest chroniony przez `oom_score_adj`, tym wyższy wynik i większa szansa na bycie ofiarą. Zabity proces to niekoniecznie ten, który spowodował wyczerpanie pamięci — to po prostu ten, który w chwili decyzji miał najgorszy wynik.

3. Masz dwa polecenia: `ps aux --sort=-%mem | head` i `smem -t -k`. Kiedy dadzą różny ranking na szczycie listy?

Odpowiedź

Gdy w grze są procesy intensywnie korzystające z bibliotek współdzielonych albo pamięci współdzielonej (np. wiele workerów tej samej aplikacji). `ps` liczy pełny `RSS` każdego procesu osobno, więc proces współdzielący dużo pamięci z innymi wygląda na cięższy, niż jest naprawdę. `smem` liczy `PSS`, dzieląc współdzieloną pamięć proporcjonalnie, więc jego ranking bliżej odzwierciedla realny wkład każdego procesu w zużycie RAM-u.

4. `journalctl -k | grep -i oom` nie zwraca nic, mimo że aplikacja padła z komunikatem podobnym do zabicia procesu. Co sprawdzisz dalej?

Odpowiedź

Czy journal jest w ogóle trwały (`Storage=persistent` w `/etc/systemd/journald.conf`) i czy zdarzenie nie sprzed poprzedniego rozruchu (`journalctl -k -b -1 | grep -i oom` sprawdza poprzedni boot). Warto też sprawdzić `dmesg -T`, jeśli maszyna nie była od tego czasu restartowana — bufor jądra i journal to dwa niezależne miejsca, które mogą się różnić zawartością zależnie od konfiguracji.

Powiązane: SWAP: czym jest, kiedy czyścić, kiedy nie ruszać · Trendy zamiast migawek: vmstat, iostat, sar, pidstat · ps — zdjęcie procesów · journalctl — dziennik systemowy

Źródła: `man 1 free` · `man 5 proc` · `man 1 smem` · `man 1 systemd-cgtop` · wewnętrzna notatka „Monitorowanie zasobów: free, vmstat, iostat, sar"

Revision #16
Created 2026-09-17 21:43:26 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:12 UTC by Tomasz Konieczny