Kto nasłuchuje i kto z kim gada: ss, lsof, netstat

◫◫ Poziom: podstawy

◫◫ Dotyczy: wszystkie dystrybucje (ss z pakietu iproute2), lsof i netstat opcjonalne

▯ Czas: ~18 min

„Co nasłuchuje na porcie 8080" i „kto z kim ma teraz nawiązane połączenie" to dwa z najczęściej zadawanych pytań diagnostycznych. `ss` odpowiada na oba, prosto z jądra; `lsof` i stary `netstat` robią to samo, tylko wolniej i z innego źródła danych.

◫◫ Skróty użyte na tej stronie (11)

Skrót	Co znaczy
CSV	w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami
HTTP	Hypertext Transfer Protocol: protokół stron WWW
HTTPS	HTTP zabezpieczony szyfrowaniem TLS
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
PID	Process ID: numer procesu w systemie
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
RST	Reset: wyprowadzenie modułu, którym płytka go restartuje, w ćwiczeniu podłączone do GPIO22
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
TCP	Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia
UDP	User Datagram Protocol: szybki protokół sieciowy bez potwierdzeń
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Fiszka

◫◫ TYLKO ODCZYT

Wypisanie kompletu nasłuchujących gniazd: ss -tulnp

```
sudo ss -tulnp
```

Klasyk administratora, odpowiednik dawnego `netstat -tunlp`: `t` (TCP), `u` (UDP), `l` (listening — tylko gniazda nasłuchujące), `n` (numeric — porty jako liczby, bez rozwiązywania nazw usług), `p` (process — dopisuje PID i nazwę procesu, wymaga uprawnień, patrz *Jak to działa*).

Przykład:

```
Netid  State   Recv-Q  Send-Q   Local Address:Port    Peer Address:Port  Process
tcp    LISTEN  0       128            0.0.0.0:22           0.0.0.0:*
users:(("sshd",pid=812,fd=3))
tcp    LISTEN  0       511            0.0.0.0:443           0.0.0.0:*
users:(("nginx",pid=4821,fd=6))
udp    UNCONN  0       0              0.0.0.0:68            0.0.0.0:*
users:(("dhclient",pid=655,fd=5))
```

◫◫ **TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia · **UDP** = User Datagram Protocol: szybki protokół sieciowy bez potwierdzeń · **PID** = Process ID: numer procesu w systemie · **ss** = socket statistics: kto nasłuchuje i jakie są połączenia

◫◫ TYLKO ODCZYT

Wypisanie wszystkich gniazd, nie tylko nasłuchujących: ss -tunap

```
sudo ss -tunap
```

Bez `l` — pokazuje też gniazda w innych stanach: `ESTAB` (nawiązane), `TIME-WAIT`, `CLOSE-WAIT`. `a` (all) to właśnie flaga rozszerzająca zakres z samego nasłuchiwania na wszystko.

Przykład:

```
Netid State      Recv-Q Send-Q  Local Address:Port   Peer Address:Port  Process
tcp   ESTAB      0      0           10.20.10.42:22   10.20.20.20:51422
users:(("sshd",pid=6011,fd=4))
tcp   TIME-WAIT  0      0          10.20.10.42:443  198.51.100.9:58212
tcp   LISTEN     0      128            0.0.0.0:22           0.0.0.0:*
users:(("sshd",pid=812,fd=3))
```

□□ **ss** = socket statistics: kto nasłuchuje i jakie są połączenia

□□ TYLKO ODCZYT

Ograniczenie wyniku do TCP nasłuchujących: ss -tlnp

```
sudo ss -tlnp
```

Zawężenie do samego TCP, gdy UDP w danej chwili nie interesuje — najczęściej przy szukaniu usługi webowej albo bazy danych.

Przykład:

```
State   Recv-Q  Send-Q   Local Address:Port    Peer Address:Port  Process
LISTEN  0       128            0.0.0.0:22           0.0.0.0:*
users:(("sshd",pid=812,fd=3))
LISTEN  0       128          127.0.0.1:5432          0.0.0.0:*
users:(("postgres",pid=1203,fd=7))
```

□□ **TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia · **UDP** = User Datagram Protocol: szybki protokół sieciowy bez potwierdzeń · **ss** = socket statistics: kto nasłuchuje i jakie są połączenia

□□ TYLKO ODCZYT

Wyświetlenie podsumowania liczbowego: ss -s

```
ss -s
```

Zbiorcze liczniki bez listy pojedynczych gniazd: ile jest w każdym stanie, ile TCP, UDP, gniazd uniksowych. Szybki „puls" systemu przed szukaniem szczegółów.

Przykład:

```
Total: 412
TCP:   68 (estab 12, closed 40, orphaned 0, timewait 38)

Transport Total     IP        IPv6
RAW□      1         0         1
UDP□      9         6         3
TCP□      28        21        7
```

```
INET□       38        27        11
FRAG□       0         0         0
```

□□ **TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia ·**UDP** = User Datagram Protocol: szybki protokół sieciowy bez potwierdzeń ·**ss** = socket statistics: kto nasłuchuje i jakie są połączenia

□□ TYLKO ODCZYT

Filtrowanie po stanie połączenia: ss -tan state established

```
ss -tan state established
```

> **Pojęcie: stan połączenia TCP.** Etykieta opisująca, w którym momencie negocjacji albo transferu znajduje się dana para gniazd TCP — od `LISTEN` (czeka na przychodzące żądania) przez `ESTABLISHED` / `ESTAB` (trwa wymiana danych) po `TIME-WAIT` i `CLOSE-WAIT` (różne fazy zamykania). Pełna tabela stanów i ich interpretacja operacyjna — w sekcji *Jak to działa*. `ss` pozwala filtrować bezpośrednio po nazwie stanu, bez ręcznego przeglądania wyniku.

Przykład:

```
State  Recv-Q Send-Q       Local Address:Port      Peer Address:Port
ESTAB  0      0               10.20.10.42:22      10.20.20.20:51422
ESTAB  0      0              10.20.10.42:443     198.51.100.9:60110
```

□□ **TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia ·**ss** = socket statistics: kto nasłuchuje i jakie są połączenia

□□ TYLKO ODCZYT

Sprawdzenie wieku gniazd w danym stanie: ss -o state time-wait

```
ss -o state time-wait
```

`-o` (options/timer) dokłada informację o liczniku czasu — ile jeszcze zostało gniazdu w bieżącym stanie, zanim jądro je zwolni. Przydaje się przy podejrzeniu, że coś generuje nietypowo dużo `TIME-WAIT`.

Przykład:

```
Netid  State      Recv-Q Send-Q      Local Address:Port     Peer Address:Port   Timer
tcp    TIME-WAIT  0      0              10.20.10.42:443    198.51.100.9:58212  timewait
(54sec)
```

ss = socket statistics: kto nasłuchuje i jakie są połączenia

TYLKO ODCZYT

Filtrowanie po adresie docelowym: ss -tp dst AdresIP

```
ss -tp dst AdresIP
```

Zawęża wynik do połączeń, których drugą stroną (z punktu widzenia lokalnej maszyny to `dst` — adres, do którego wychodzi ruch) jest konkretny host. Odwrotnością jest `src`, filtrujące po adresie lokalnym.

Przykład:

```
State  Recv-Q Send-Q     Local Address:Port    Peer Address:Port  Process
ESTAB  0      0           10.20.20.20:44210      10.20.10.42:443
users:(("curl",pid=9931,fd=5))
```

ss = socket statistics: kto nasłuchuje i jakie są połączenia

TYLKO ODCZYT

Wypisanie gniazd uniksowych: ss -x

```
ss -x
```

Gniazda domeny uniksowej (`AF_UNIX`) — komunikacja między procesami na tej samej maszynie przez plik w systemie plików, bez udziału stosu sieciowego IP. Przykłady: gniazdo, przez które `systemd` rozmawia z `journald`, albo `/var/run/docker.sock`.

Przykład:

```
Netid  State    Recv-Q  Send-Q   Local Address:Port                    Peer Address:Port
u_str  LISTEN   0       4096     /run/docker.sock 21044                          * 0
u_str  ESTAB    0       0        /run/systemd/journal/stdout 15872               * 0
```

IP = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **ss** = socket statistics: kto nasłuchuje i jakie są połączenia

TYLKO ODCZYT

Wypisanie nasłuchujących portów TCP od strony procesów: lsof -nP -iTCP -sTCP:LISTEN

```
sudo lsof -nP -iTCP -sTCP:LISTEN
```

Inne narzędzie, ten sam cel co pierwsza komenda tej strony: `-n` wyłącza rozwiązywanie nazw hostów, `-P` wyłącza rozwiązywanie nazw portów na nazwy usług (`443` zamiast `https`), `-iTCP` ogranicza do gniazd TCP, `-sTCP:LISTEN` filtruje po stanie. Wolniejsze niż `ss`, bo `lsof` przechodzi przez deskryptory plików każdego procesu — ale bywa czytelniejsze przy szukaniu „który dokładnie proces trzyma ten port".

Przykład:

```
COMMAND   PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
sshd      812  root    3u  IPv4  18234      0t0  TCP *:22 (LISTEN)
nginx    4821  root    6u  IPv4  28841      0t0  TCP *:443 (LISTEN)
postgres 1203  postgres 7u IPv4  30120      0t0  TCP 127.0.0.1:5432 (LISTEN)
```

TCP = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia

TYLKO ODCZYT

Sprawdzenie, kto trzyma konkretny port: lsof -i :NumerPortu

```
sudo lsof -i :NumerPortu
```

Najszybsze pytanie „czy ktoś już siedzi na porcie 8080, zanim tam odpalę swoją usługę" — bez filtrowania po stanie ani protokole, `lsof` pokaże wszystko związane z tym numerem portu.

Przykład:

```
COMMAND   PID     USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
python3 22011 ola         3u  IPv4  55210      0t0  TCP *:8080 (LISTEN)
```

TYLKO ODCZYT

Wypisanie wszystkich gniazd jednego procesu: lsof -p NumerPID -i

```
sudo lsof -p NumerPID -i
```

Odwrotny kierunek pytania: nie „kto jest na porcie X", tylko „co dokładnie robi w sieci ten konkretny proces" — wszystkie jego połączenia i nasłuchy naraz.

Przykład:

```
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
nginx   4821 root    6u  IPv4  28841      0t0  TCP *:443 (LISTEN)
nginx   4821 root    7u  IPv4  31402      0t0  TCP 10.20.10.42:443->10.20.20.20:51680
(ESTABLISHED)
```

□□ TYLKO ODCZYT

Wypisanie nasłuchujących gniazd starszym narzędziem: netstat -tunlp

```
sudo netstat -tunlp
```

Ta sama informacja co pierwsza fiszka tej strony, z pakietu `net-tools`. Działa, ale narzędzie jest **wycofane** od lat na rzecz `ss` — nowsze minimalne instalacje RHEL/Rocky 8+ nie mają go domyślnie. Warto rozpoznawać składnię (bo mnóstwo starych poradników i skryptów wciąż jej używa), ale w nowych skryptach używamy `ss`.

Przykład:

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address     Foreign Address   State    PID/Program name
tcp        0      0 0.0.0.0:22        0.0.0.0:*         LISTEN   812/sshd
tcp        0      0 0.0.0.0:443       0.0.0.0:*         LISTEN   4821/nginx
udp        0      0 0.0.0.0:68        0.0.0.0:*                  655/dhclient
```

□□ **RHEL** = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

□□ ZMIENIA SYSTEM

Skrypt: eksport aktywnych połączeń z nazwą procesu do CSV

```
csvPath="/tmp/connections.csv"
```

```
netstat -tunp | awk '
BEGIN { print "LocalAddress:Port,RemoteAddress:Port,ProcessName" }
$1 == "tcp" {
    split($4, local, ":")
    split($5, remote, ":")
    pid = $7
    if (pid ~ /[0-9]+\/.*/) {
        split(pid, proc, "/")
        cmd = "ps -p " proc[1] " -o comm="
        cmd | getline processName
        close(cmd)
        print local[1] ":" local[2] "," remote[1] ":" remote[2] "," processName
    }
}' | tee "$csvPath"

echo "Zapisano połączenia do pliku CSV: $csvPath"
```

Starszy, sprawdzony w produkcji skrypt do raportu „kto z kim rozmawia i jakim procesem" — bierze surowe wyjście `netstat -tunp`, rozbija kolumny lokalnego i zdalnego adresu na adres/port, a numer PID doklejony przez `netstat` do nazwy procesu (`4821/nginx`) rozdziela ukośnikiem i dociąga pełną nazwę procesu przez `ps -p PID -o comm=`. Wynik ląduje jednocześnie na ekranie i w pliku CSV dzięki `tee`. Współczesny odpowiednik korzystałby z `ss -tunp` zamiast `netstat` — kolumny mają inny układ, ale ten sam pomysł (rozbicie `adres:port`, wyciągnięcie PID z pola procesu) przenosi się wprost.

Przykład:

```
LocalAddress:Port,RemoteAddress:Port,ProcessName
10.20.10.42:22,10.20.20.20:51422,sshd
10.20.10.42:443,198.51.100.9:60110,nginx
Zapisano połączenia do pliku CSV: /tmp/connections.csv
```

□□ **CSV** = w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami · **PID** = Process ID: numer procesu w systemie · **split** = podziel duży plik na mniejsze części o wskazanym rozmiarze

□□ **TYLKO ODCZYT**

Sprawdzenie, który proces trzyma port (z opcją zabicia): fuser -n tcp NumerPortu

```
sudo fuser -n tcp NumerPortu
```

Krótsza forma pytania „czyj to port": wypisuje sam PID, bez żadnych dodatkowych kolumn. Dopisanie `-k` (`sudo fuser -k -n tcp NumerPortu`) od razu zabija proces trzymający port — wygodne w skryptach czyszczących, ale nieodwracalne bez potwierdzenia.

Przykład:

```
8080/tcp:             22011
```

PID = Process ID: numer procesu w systemie

TYLKO ODCZYT

Zamknięcie pasujących gniazd bez zabijania procesu: ss -K

```
sudo ss -K dst AdresIP dport = NumerPortu
```

Zrywa konkretne połączenie na poziomie gniazda TCP (wysyłając RST), nie dotykając samego procesu — inaczej niż `fuser -k`, który zabija cały proces. Przydatne, gdy chcemy uciąć jedno zawieszone połączenie klienta, a usługa ma zostać uruchomiona dalej.

Przykład:

```
Netid State      Recv-Q Send-Q     Local Address:Port     Peer Address:Port
tcp   ESTAB      0      0             10.20.10.42:443    198.51.100.9:58212
```

Wyjście `ss -K` pokazuje gniazda, które faktycznie zamknięto, w tym samym formacie co zwykłe `ss`.

TCP = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia ·**RST** = Reset: wyprowadzenie modułu, którym płytka go restartuje, w ćwiczeniu podłączone do GPIO22 ·**ss** = socket statistics: kto nasłuchuje i jakie są połączenia

TYLKO ODCZYT

Podejrzenie surowych danych źródłowych: /proc/net/tcp

```
cat /proc/net/tcp | head -3
```

To jest plik, który parsuje `netstat` (i który `ss` czyta znacznie efektywniej przez interfejs `netlink`, patrz *Jak to działa*). Adresy i porty są zapisane w postaci szesnastkowej, w

kolejności bajtów sieci — dlatego surowy odczyt tego pliku bez narzędzia jest praktycznie nieczytelny na oko.

Przykład:

```
  sl  local_address rem_address   st tx_queue rx_queue tr tm->when retrnsmt   uid
timeout inode
   0: 00000000:0016 00000000:0000 0A 00000000:00000000 00:00000000 00000000
0        0 18234
   1: 2A0A140A:01BB 00000000:0000 0A 00000000:00000000 00:00000000 00000000
0        0 28841
```

`0016` po rozbiciu na dziesiętne to port 22 (`0x16`), `01BB` to port 443 (`0x1BB`) — szesnastkowo, tak jak zapisuje je jądro w tym pliku.

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Wszystkie flagi `ss`

Flaga	Co robi
`-t`	TCP
`-u`	UDP
`-l`	tylko gniazda nasłuchujące (LISTEN)
`-a`	wszystkie gniazda, nie tylko nasłuchujące
`-n`	adresy i porty jako liczby, bez rozwiązywania nazw
`-p`	dopisz PID i nazwę procesu (wymaga uprawnień)
`-o`	dopisz licznik czasu/timer stanu
`-x`	gniazda uniksowe (`AF_UNIX`)
`-s`	podsumowanie liczbowe zamiast listy
`-K`	zamknij pasujące gniazda (RST)
`state nazwa`	filtr po stanie (`established`, `time-wait`, `listening`…)
`src` / `dst`	filtr po adresie lokalnym/zdalnym

Uważaj

- Bez `sudo`/roota kolumna `Process` w `ss -p` i `lsof` bywa pusta dla cudzych procesów — to nie błąd narzędzia, tylko brak uprawnień do zajrzenia w ich deskryptory.
- `netstat` działa, ale jest wycofany — nowe skrypty pisz na `ss`.
- `fuser -k` zabija **cały proces**, nie samo połączenie; do zerwania pojedynczego połączenia bez ubijania usługi służy `ss -K`.

PID = Process ID: numer procesu w systemie · **RST** = Reset: wyprowadzenie modułu, którym płytka go restartuje, w ćwiczeniu podłączone do GPIO22

Z życia wzięte

Z ŻYCIA WZIĘTE

Piątek, 16:00. Aplikacja zespołu Oli na app01 nie chce wystartować, bo port 8080 jest podobno zajęty. Tomek sprawdza komendą ss, który proces faktycznie trzyma ten port, i okazuje się, że to poprzednia wersja aplikacji wisi w stanie nasłuchiwania po nieudanym wcześniejszym restarcie.

Po co to istnieje

Pojęcie: gniazdo (socket). Punkt końcowy komunikacji sieciowej wewnątrz jądra — obiekt, który proces otwiera, żeby wysyłać i odbierać dane przez sieć albo lokalnie między procesami. Gniazdo TCP/UDP jest zawsze opisane parą: adres IP i numer portu. Wszystkie narzędzia tej strony robią jedno i to samo: odpytują jądro o listę aktualnie istniejących gniazd i to, co o nich wiadomo — kto je otworzył, w jakim są stanie, z kim rozmawiają.

Pojęcie: port. Liczba od 1 do 65535 identyfikująca konkretną usługę na danym adresie IP — bez portu maszyna wiedziałaby tylko, że przyszedł jakiś ruch, ale nie do którego z kilkudziesięciu jednocześnie działających programów go skierować. Port 22 to zwyczajowo SSH, 443 HTTPS, 5432 PostgreSQL — to tylko konwencja (`/etc/services`), nic nie stoi na przeszkodzie, by uruchomić dowolną usługę na dowolnym wolnym porcie.

Pytanie „co nasłuchuje na tym porcie" i „kto z kim aktualnie rozmawia" wraca bez końca w codziennej pracy: nowa usługa nie startuje, bo port jest zajęty; podejrzenie, że coś nawiązuje połączenia, których nie powinno; audyt, które usługi w ogóle są wystawione na zewnątrz. `ss`, `lsof`

i stary `netstat` to trzy narzędzia odpowiadające na te same pytania, różniące się tym, skąd biorą dane i jak szybko to robią — różnica opisana w *Jak to działa*.

Analogia

Wyobraź sobie **centralę telefoniczną dużej firmy** z setką numerów wewnętrznych.

Gniazdo nasłuchujące to telefon, który leży odłożony, ale gotowy — sekretarka przy numerze wewnętrznym 22 czeka, aż ktoś zadzwoni, i nikt jeszcze nie rozmawia. To jest `LISTEN`. Gniazdo nawiązane (`ESTABLISHED`) to trwająca rozmowa — dwie strony, słuchawki podniesione, wymiana informacji w toku. `TIME-WAIT` to moment tuż po odłożeniu słuchawki, gdy linia jeszcze przez chwilę „się wygrzewa" — system celowo trzyma ją zarezerwowaną, żeby żadna spóźniona wiadomość z zakończonej już rozmowy nie trafiła przypadkiem do nowej. `CLOSE-WAIT` to sytuacja odwrotna i dużo bardziej niepokojąca: druga strona już dawno odłożyła słuchawkę, a Twoja centralka o tym nie wie i wciąż trzyma linię otwartą — bo nikt (czyli aplikacja) nie kazał jej zamknąć gniazda.

`ss` to podgląd wprost z centralki cyfrowej — zapytanie do samego systemu komutacyjnego, natychmiastowe. `lsof` i `netstat` to obchodzenie biurek jedno po drugim i pytanie każdego pracownika osobno, czy trzyma słuchawkę — dokładne, ale wolniejsze przy stu biurkach.

Jak to działa

Rozwiń: jak to działa pod spodem

`ss` (socket statistics) czyta stan gniazd przez interfejs jądra `sock_diag` (rodzina `netlink`, podobnie jak `ip` z pierwszej strony rozdziału) — binarny, zaprojektowany specjalnie do masowego odpytywania o gniazda, bez kosztu parsowania tekstu. To źródło danych jest dokładnie tym, co widać w `/proc/net/tcp` i `/proc/net/udp`, tylko podane w dużo tańszej do przetworzenia formie.



Gniazdo TCP przechodzi kolejne stany od nasłuchiwania po zamknięcie połączenia. (kliknięcie otwiera pełny rozmiar)

`netstat` z kolei **czyta wprost te pliki tekstowe** w `/proc/net/` i sam je parsuje w przestrzeni użytkownika, linia po linii, w formacie z adresami szesnastkowo zakodowanymi w kolejności bajtów sieciowych (`0016` = port 22). Przy dużej liczbie gniazd — tysiącach połączeń na zajętym serwerze WWW — różnica prędkości między `ss` a `netstat` jest wyraźnie mierzalna, i to jest jeden z powodów, dla którego `net-tools` (skąd pochodzi `netstat`) jest od lat uznawane za przestarzałe i nie instalowane domyślnie na RHEL/Rocky 8 i nowszych.

`lsof` idzie jeszcze inną drogą: przechodzi przez katalog `/proc/PID/fd/` **każdego** procesu w systemie, sprawdza, które z jego deskryptorów są gniazdami sieciowymi (dowiązania w stylu `socket:[28841]`), i dopiero na tej podstawie składa listę. To wyjaśnia, dlaczego `lsof` jest wolniejszy niż `ss` przy szerokim zapytaniu (`lsof -i` bez filtra), ale bywa wygodniejszy przy pytaniu odwrotnym — „co robi w sieci ten jeden konkretny proces" (`lsof -p PID -i`) — bo naturalnie operuje na perspektywie procesu, nie gniazda.

Dlaczego `-p` wymaga roota. Skojarzenie gniazda z konkretnym PID-em i nazwą procesu wymaga zajrzenia do tabeli deskryptorów **cudzego** procesu — a to jest operacja, którą jądro pozwala wykonać tylko właścicielowi procesu albo rootowi (ten sam mechanizm uprawnień, co przy `/proc/PID/fd/` opisanym na stronie o `ps`). Uruchomienie `ss -tlnp` bez `sudo` pokaże wszystkie gniazda, ale kolumnę `Process` wypełni tylko dla własnych procesów — dla cudzych zostanie pusta, bez żadnego komunikatu błędu, co bywa mylące.

> **Pojęcie: gniazdo nasłuchujące kontra nawiązane.** Gniazdo nasłuchujące (`LISTEN`) to punkt, w którym proces zadeklarował gotowość do przyjmowania nowych połączeń na danym porcie — samo w sobie nie przesyła żadnych danych aplikacji. Gniazdo nawiązane (`ESTABLISHED`) powstaje dopiero po udanym trójetapowym uzgadnianiu TCP (SYN, SYN-ACK, ACK) i to właśnie przez nie płyną rzeczywiste dane. Jeden proces może mieć jedno gniazdo `LISTEN` na porcie 443 i jednocześnie setki gniazd `ESTABLISHED` — po jednym na każdego aktualnie podłączonego klienta.

Tabela stanów TCP i ich interpretacja operacyjna.

Stan	Co znaczy	Interpretacja operacyjna
`LISTEN`	proces czeka na nowe połączenia na tym porcie	normalne — brak `LISTEN` na oczekiwanym porcie znaczy, że usługa nie wystartowała albo nasłuchuje gdzie indziej
`ESTABLISHED`	połączenie w pełni nawiązane, trwa transfer danych	normalne, liczba rośnie i maleje z obciążeniem
`TIME-WAIT`	strona, która pierwsza zamknęła połączenie, czeka na wygaśnięcie okna (zwykle ok. 60 s), zanim zwolni krotkę adres:port	**normalne w dużej liczbie** przy serwerach obsługujących wiele krótkich połączeń (np. HTTP bez keep-alive) — mechanizm ochronny TCP, nie błąd

Stan	Co znaczy	Interpretacja operacyjna
`CLOSE-WAIT`	zdalna strona zamknęła połączenie, ale lokalna aplikacja jeszcze nie wywołała `close()` na swoim gnieździe	**duża i rosnąca liczba to sygnał błędu w aplikacji** — proces nie zwalnia gniazd po zakończonej rozmowie, co prowadzi do wyczerpania deskryptorów plików

Rozróżnienie „dużo `TIME-WAIT` to normalne, dużo `CLOSE-WAIT` to błąd" jest jednym z najczęściej mylonych faktów w diagnostyce sieci — oba stany brzmią podobnie („coś czeka"), ale mówią o zupełnie różnych rzeczach: pierwszy o higienie protokołu po Twojej stronie, drugi o zaniedbaniu po stronie Twojej aplikacji.

Krok po kroku

1. Zrób ogólny przegląd tego, co nasłuchuje.

```
sudo ss -tulnp
```

2. Sprawdź podsumowanie liczbowe, żeby ocenić skalę problemu, zanim zaczniesz szukać szczegółów.

```
ss -s
```

```
TCP:   612 (estab 40, closed 520, orphaned 0, timewait 512)
```

612 gniazd TCP, z czego 512 w `TIME-WAIT` — przy serwerze WWW obsługującym tysiące krótkich żądań na minutę to zwykle norma, nie alarm.

3. Zawęź do stanu, który faktycznie wymaga uwagi.

```
ss -tan state close-wait | wc -l
```

```
47
```

47 gniazd w `CLOSE-WAIT` to już sygnał do sprawdzenia, który proces je trzyma.

4. Znajdź winowajcę.

```
ss -tanp state close-wait | head -3
```

```
State      Recv-Q Send-Q     Local Address:Port    Peer Address:Port   Process
CLOSE-WAIT 1      0            10.20.10.42:8443  198.51.100.44:51022
users:(("javaapp",pid=15021,fd=88))
```

5. Potwierdź niezależnym narzędziem.

```
sudo lsof -p 15021 -i | grep CLOSE_WAIT | wc -l
```

```
47
```

Zgodność dwóch niezależnych narzędzi (`ss` i `lsof`) potwierdza, że to nie artefakt jednego z nich — proces `javaapp` faktycznie nie zamyka gniazd po stronie klienta.

Częste błędy

Uruchomienie `ss -p` bez uprawnień i puste `Process`.

```
$ ss -tlnp
State  Recv-Q Send-Q  Local Address:Port  Peer Address:Port
LISTEN 0      128          0.0.0.0:443         0.0.0.0:*
```

Bez `sudo` jądro nie pozwala odczytać, do którego procesu należy cudze gniazdo — kolumna `Process` po prostu znika, bez komunikatu o braku uprawnień. Naprawa: `sudo ss -tlnp`.

`netstat: command not found` na świeżym minimalnym RHEL.

```
$ netstat -tunlp
bash: netstat: command not found
```

Pakiet `net-tools` nie jest już domyślnie instalowany. Naprawa: `ss -tulnp` robi dokładnie to samo, bez doinstalowywania niczego; jeśli koniecznie potrzebny jest sam `netstat` (stary skrypt), `dnf install net-tools`.

Odczytanie dużej liczby `TIME-WAIT` jako awarii.

```
ss -s
```

```
TCP:   2140 (estab 30, closed 2000, orphaned 0, timewait 2000)
```

Panika „coś zalewa serwer połączeniami" na widok dwóch tysięcy `TIME-WAIT` przy serwerze WWW obsługującym tysiące krótkich żądań HTTP na minutę jest zwykle nieuzasadniona — to normalny

efekt uboczny zamykania połączeń zgodnie ze specyfikacją TCP. Sygnałem realnego problemu byłoby wyczerpanie puli portów efemerycznych (błędy `bind: Address already in use` przy nowych połączeniach wychodzących), nie sama liczba `TIME-WAIT`.

Zignorowanie rosnącej liczby `CLOSE-WAIT`.

```
watch -n 5 "ss -tan state close-wait | wc -l"
```

```
Every 5.0s: ss -tan state close-wait | wc -l


312
```

Liczba rośnie z każdym odświeżeniem zamiast się stabilizować — to nie jest normalny szum, tylko wyciek deskryptorów po stronie aplikacji, która nie wywołuje `close()` po zakończonych rozmowach przez drugą stronę. Naprawa nie leży w warstwie sieci — trzeba poprawić aplikację (albo, doraźnie, zrestartować proces, żeby zwolnić zablokowane gniazda), a `ss -tanp state close-wait` pokazuje, który proces jest winowajcą.

`fuser -k` zamiast `ss -K` tam, gdzie chodziło tylko o jedno połączenie.

```
sudo fuser -k -n tcp 443
```

To polecenie zabija **cały proces** nasłuchujący na porcie 443 (np. całego `nginx`), nie jedno zawieszone połączenie klienta — skutek uboczny to przerwa w działaniu usługi dla wszystkich innych klientów. Naprawa: do zerwania konkretnego połączenia bez ubijania usługi służy `ss -K dst <adres> dport = <port>`, które zamyka tylko pasujące gniazdo.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz w katalogu `/tmp`, na porcie, który nie koliduje z żadną realną usługą na maszynie. Nic nie wymaga uprawnień administratora poza jednym poleceniem wyraźnie oznaczonym `sudo`.

Ćwiczenie 1 — znajdź własny proces trzema narzędziami

```
cd /tmp
python3 -m http.server 8080 &
```

```
PID=$!

ss -tlnp | grep 8080
lsof -i :8080
fuser -n tcp 8080

kill $PID
```

Czy PID zwrócony przez wszystkie trzy narzędzia jest identyczny? Które z nich wymagało `sudo`, a które nie, i dlaczego?

Odpowiedź

PID powinien być identyczny wszędzie — trzy narzędzia patrzą na ten sam stan jądra, tylko z różnych źródeł danych (netlink dla `ss`, `/proc/PID/fd` dla `lsof` i `fuser`). Żadne nie wymaga `sudo` dla **własnego** procesu — uprawnienia do odczytu deskryptorów własnych procesów ma zawsze ich właściciel. `sudo` staje się konieczne dopiero przy sprawdzaniu procesów należących do innych użytkowników (np. `nginx` działającego jako `root`).

Ćwiczenie 2 — obejrzyj `TIME-WAIT` na żywo

```
cd /tmp
python3 -m http.server 8080 &
PID=$!

for i in 1 2 3 4 5; do curl -s -o /dev/null http://127.0.0.1:8080/; done
ss -tan state time-wait | grep 8080

kill $PID
```

Ile gniazd w `TIME-WAIT` widzisz i po jakim mniej więcej czasie znikną same, bez żadnej Twojej ingerencji?

Odpowiedź

Każde krótkie żądanie `curl` bez keep-alive kończy się nowym gniazdem w `TIME-WAIT` po stronie, która pierwsza zamknęła połączenie — zwykle widać po jednym wpisie na każde wywołanie `curl`. Domyślny czas życia stanu `TIME-WAIT` w Linuksie to ok. 60 sekund (`net.ipv4.tcp_fin_timeout` wpływa na pokrewny, ale osobny licznik) — po tym czasie jądro samo

zwalnia gniazdo bez żadnej akcji administratora. To jest normalne zachowanie protokołu, nie coś do „naprawiania".

Pytania kontrolne

1. Dlaczego duża liczba gniazd w `CLOSE-WAIT` jest bardziej niepokojąca niż duża liczba w `TIME-WAIT`?

Odpowiedź

`TIME-WAIT` to zaplanowany, krótkotrwały etap zamykania połączenia po stronie protokołu TCP — rośnie naturalnie przy dużym ruchu i sam się rozlicza. `CLOSE-WAIT` znaczy, że druga strona już zamknęła połączenie, a lokalna aplikacja nie wywołała `close()` na swoim gnieździe — to wina programu, nie protokołu, i liczba takich gniazd rośnie bez ograniczenia, aż zabraknie deskryptorów plików.

2. Dlaczego `ss -tlnp` uruchomione bez `sudo` może pokazać pustą kolumnę `Process` zamiast błędu?

Odpowiedź

Bo jądro po cichu odmawia dostępu do informacji o właścicielu gniazda należącego do procesu innego użytkownika — bez podniesionych uprawnień `ss` po prostu nie ma czego wpisać w tę kolumnę, więc zostawia ją pustą zamiast zgłaszać błąd. Dla własnych procesów kolumna wypełnia się normalnie nawet bez `sudo`.

3. Czym różni się źródło danych `ss` od źródła danych `netstat`, i dlaczego to ma znaczenie przy dużej liczbie połączeń?

Odpowiedź

`ss` odpytuje jądro bezpośrednio przez binarny interfejs `netlink`/`sock_diag`. `netstat` czyta i parsuje w przestrzeni użytkownika pliki tekstowe `/proc/net/tcp` i `/proc/net/udp`. Przy tysiącach gniazd różnica w koszcie przetworzenia jest wyraźnie odczuwalna — `ss` skaluje się lepiej, co jest jednym z powodów, dla których zastąpił `netstat` jako narzędzie domyślne.

4. Masz dwa polecenia: `fuser -k -n tcp 443` i `ss -K dst 198.51.100.9 dport = 443`. Czym różni się ich skutek dla usługi nasłuchującej na porcie 443?

Odpowiedź

`fuser -k` zabija cały proces trzymający port 443 — usługa przestaje działać dla wszystkich klientów. `ss -K` zamyka tylko gniazda pasujące do podanego filtra (tu: połączenie z konkretnym adresem zdalnym) — usługa działa dalej, traci tylko to jedno połączenie.

Powiązane: ip: adresy, trasy, interfejsy · NetworkManager i nmcli: konfiguracja trwała · firewalld: strefy, porty, usługi, rich rules · ps: zdjęcie stanu procesów

Źródła: `man 8 ss` · `man 8 lsof` · `man 8 netstat` · `man 1 fuser` · `man 5 proc` · OneNote „Ogolne" (Sprawdzenie nasłuchiwanych portów; Listowanie otwartych połączeń wraz z procesami)

Revision #16
Created 2026-09-17 21:43:17 UTC by Tomasz Konieczny
Updated 2026-10-03 19:08:59 UTC by Tomasz Konieczny