Gdy df i du nie zgadzają się: usunięte otwarte pliki

◫◫ Poziom: średniozaawansowany

◫◫ Dotyczy: wszystkie dystrybucje (mechanizm jest częścią jądra Linux, nie konkretnej dystrybucji)

⏱ Czas: ~18 min

`df -h` mówi, że dysk jest pełny. `du -sh` przeszukuje ten sam dysk i nie znajduje niczego, co by to tłumaczyło. To nie usterka narzędzi — to usunięty plik, którego wciąż trzyma otwartym jakiś proces.

◫◫ Skróty użyte na tej stronie (5)

Skrót	Co znaczy
DNS	Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
PID	Process ID: numer procesu w systemie
POSIX	Portable Operating System Interface: wspólny zestaw zasad, dzięki któremu polecenie działa tak samo na różnych Uniksach

Fiszka

◫◫ TYLKO ODCZYT

Sprawdzenie punktu wyjścia: df -h

```
df -h /var
```

Pierwszy sygnał alarmowy. Zapamiętujemy dokładną liczbę `Used` — do niej będziemy porównywać wynik po naprawie.

Przykład:

```
Filesystem                 Size  Used Avail Use% Mounted on
/dev/mapper/vgVAR-lvVAR        30G   29G  380M  99% /var
```

df = disk free: ile miejsca wolnego na dyskach

ZMIENIA SYSTEM

Podsumowanie katalogów w /var: du -sh

```
du -sh /var/* 2>/dev/null | sort -rh
```

Suma wyniku wychodzi wyraźnie mniejsza niż `Used` z `df -h`. To jest moment, w którym administrator, który nie zna tego mechanizmu, zaczyna podejrzewać uszkodzony system plików — a przyczyna jest dużo prostsza.

Przykład:

```
9.4G    /var/lib
6.1G    /var/log
1.2G    /var/cache
16G     (suma widocznych plików)
```

16G widocznych plików wobec 29G zgłaszanych przez `df`. Różnica — 13G — leży gdzieś, czego `du` nie widzi.

du = disk usage: ile miejsca zajmuje katalog

TYLKO ODCZYT

Wyszukanie usuniętych, wciąż otwartych plików: +L1

```
sudo lsof +L1
```

`+L1` każe `lsof` wypisać tylko te otwarte pliki, których licznik dowiązań (`link count`) spadł poniżej 1 — czyli pliki, które ktoś usunął (`rm`, `logrotate`, aplikacja czyszcząca stare dane), ale które wciąż ma otwarte jakiś proces. To jest dokładnie ten rodzaj pliku, którego `du` nigdy nie znajdzie, bo `du` przegląda katalogi, a plik bez nazwy w żadnym katalogu już nie istnieje.

Przykład:

```
COMMAND    PID   USER   FD   TYPE DEVICE SIZE/OFF NLINK   NODE NAME
nginx     4821   nginx   6w   REG  253,1  13958643712     0 524301
/var/log/nginx/access.log (deleted)
```

Kolumna `NLINK` równa `0` i dopisek `(deleted)` po nazwie to jednoznaczny odcisk palca tego problemu. Kolumna `SIZE/OFF` pokazuje niemal 13 GB — dokładnie tyle, ile brakowało w rachunku `du`.

GB = Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej

TYLKO ODCZYT

Zawężenie wyszukiwania, bez rozwiązywania nazw: -nP +L1

```
sudo lsof -nP +L1 /var
```

`-n` wyłącza rozwiązywanie adresów sieciowych na nazwy hostów, `-P` wyłącza rozwiązywanie numerów portów na nazwy usług — bez nich `lsof` na maszynie z wieloma połączeniami sieciowymi potrafi wisieć długie sekundy, próbując odpytać DNS dla każdego gniazda. Argument `/var` zawęża wynik do plików leżących na systemie plików zamontowanym pod `/var`, więc od razu widzimy tylko to, co dotyczy zapełnionego wolumenu.

Przykład:

```
COMMAND    PID   USER   FD   TYPE DEVICE SIZE/OFF NLINK   NODE NAME
nginx     4821   nginx   6w   REG  253,1  13958643712     0 524301
/var/log/nginx/access.log (deleted)
```

DNS = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP

ZMIENIA SYSTEM

Wyszukanie usuniętych plików starszym sposobem: lsof | grep deleted

```
sudo lsof / 2>/dev/null | grep deleted
```

Zanim `lsof` miał flagę `+L1`, administratorzy szukali dopisku `(deleted)` w pełnym wyniku przez `grep`. Działa wszędzie, ale jest wolniejsze (przegląda wszystkie otwarte pliki w systemie, nie tylko te z zerowym licznikiem dowiązań) i mniej precyzyjne — złapie też np. usunięte pliki tymczasowe, które i tak zaraz same się zamkną.

Przykład:

```
nginx     4821   nginx    6w      REG              253,1 13958643712        0  524301
/var/log/nginx/access.log (deleted)
```

TYLKO ODCZYT

Sprawdzenie usuniętych plików po znanym PID: ls -l /proc/PID/fd

```
sudo ls -l /proc/4821/fd | grep deleted
```

Jeśli `lsof` z jakiegoś powodu nie jest dostępny (minimalny obraz kontenera), a numer PID podejrzanego procesu jest już znany — z `ps` albo z poprzedniego kroku — ten sam fakt sprawdzamy bezpośrednio w `/proc`, bez dodatkowego pakietu.

Przykład:

```
lrwx------ 1 nginx nginx 64 lis 20 09:41 6 -> /var/log/nginx/access.log (deleted)
```

Liczba `6` to numer deskryptora — potrzebny w następnym kroku.

PID = Process ID: numer procesu w systemie · **ls** = list: wypisz zawartość katalogu

OSTROŻNIE · NIEODWRACALNE

Zwolnienie miejsca bez restartu: truncate -s 0

```
sudo truncate -s 0 /proc/4821/fd/6
```

To jest sedno naprawy bez przestoju. Deskryptor `/proc/4821/fd/6` to dowiązanie do tego samego inode'u, który proces trzyma otwarty — obcięcie go do zera bajtów zwalnia całą przestrzeń zajmowaną przez usunięty plik, a proces **nadal ma ten sam, ważny deskryptor** i po prostu zaczyna dopisywać od nowego, pustego początku. Usługa nie zauważa przerwy, bo z jej punktu widzenia deskryptor cały czas wskazuje na coś, do czego może pisać.

□□ ZMIENIA SYSTEM

Czyste rozwiązanie restartem usługi: systemctl restart

```
sudo systemctl restart nginx
```

Restart zamyka wszystkie stare deskryptory procesu (w tym ten trzymający usunięty plik — jądro w tym momencie faktycznie zwalnia blok na dysku) i uruchamia proces od nowa, który otwiera log pod jego właściwą, aktualną nazwą. To rozwiązanie czystsze niż `truncate` na żywym deskryptorze, ale wymaga chwilowej przerwy w działaniu usługi — stąd `truncate` jako doraźny ratunek, gdy przestój nie wchodzi w grę.

□□ **systemctl** = system control: sterowanie usługami systemd

□□ TYLKO ODCZYT

Pominięcie zasobów zamontowanych pod spodem: --exclude

```
du -sh --exclude='lost+found' /var/lib/app
```

Gdy pod liczonym katalogiem jest zamontowany kolejny system plików albo katalog specjalny (`lost+found` po `fsck`), `--exclude` pozwala pominąć konkretny wzorzec bez opuszczania całej gałęzi przez `-x`. Przydatne, gdy chcemy policzyć realną zawartość katalogu, a nie techniczne artefakty systemu plików.

□□ **du** = disk usage: ile miejsca zajmuje katalog

□□ TYLKO ODCZYT

Sprawdzenie zamontowań pod katalogiem: -R

```
findmnt -R /var
```

`-R` (*recursive*) pokazuje wszystkie punkty montowania w poddrzewie zaczynającym się od podanej ścieżki — nie tylko `/var` jako całość, ale też ewentualny osobny wolumen zamontowany głębiej, np. `/var/lib/docker`. To pierwsze pytanie, zanim zaczniemy podejrzewać usunięte pliki: może `du /var` po prostu nie liczy czegoś, co jest osobnym punktem montowania z inną logiką rotacji logów.

Przykład:

```
TARGET              SOURCE                     FSTYPE OPTIONS
/var                /dev/mapper/vgVAR-lvVAR    xfs    rw,relatime
└─/var/lib/docker   /dev/mapper/vgDocker-lvDocker xfs    rw,relatime
```

findmnt = pokaż zamontowane systemy plików w czytelnej formie drzewa

TYLKO ODCZYT

Potwierdzenie naprawy: df -h

```
df -h /var
```

Przykład:

```
Filesystem                 Size  Used Avail Use% Mounted on
/dev/mapper/vgVAR-lvVAR        30G   16G   13G  56% /var
```

`Used` spadło z 29G do 16G — dokładnie o te ~13G, które trzymał usunięty, ale wciąż otwarty plik. To jest dowód, że diagnoza była poprawna, nie tylko poprawa „na oko".

df = disk free: ile miejsca wolnego na dyskach

TYLKO ODCZYT

Wszystkie flagi

Komenda	Co pokazuje
`df -h` (przed i po)	czy problem rzeczywiście zniknął, w liczbach
`du -sh`, `du -sh --exclude`	co widać po nazwach plików w katalogach
`lsof +L1`, `lsof -nP +L1 ŚCIEŻKA`	pliki z zerowym licznikiem dowiązań, wciąż otwarte

Komenda	Co pokazuje
`lsof | grep deleted`	to samo, starszym, uniwersalnym sposobem

Komenda	Co pokazuje
`ls -l /proc/PID/fd \	grep deleted`
`truncate -s 0 /proc/PID/fd/N`	zwolnienie miejsca bez przerywania pracy procesu
`systemctl restart usluga`	czyste zamknięcie starego deskryptora, z przestojem
`findmnt -R`	czy pod katalogiem nie kryje się osobny punkt montowania

to samo dla jednego znanego procesu, bez `lsof`

Uważaj

- `truncate` działa na deskryptorze konkretnego procesu — jeśli plik ma otwarty więcej niż jeden proces (rzadkie, ale zdarza się przy workerach współdzielących log), trzeba obciąć każdy z nich albo po prostu zrestartować

usługę.

- Obcięcie **złego** deskryptora (aktywnego, nieusuniętego pliku) kasuje dane usługi bez ostrzeżenia — zawsze weryfikuj `NLINK`/`(deleted)` przed `truncate`.
- Samo `rm` na pliku, który już jest usunięty przez inny proces (np. logrotate skasował go już wcześniej), nic nie zmieni — miejsce i tak zwolni się dopiero, gdy ostatni proces zamknie swój deskryptor.

Z życia wzięte

Z ŻYCIA WZIĘTE

Piątek, 16:00. Pani Halina zgłasza, że Optima znowu wolno chodzi, a Tomek widzi na sql01 coś dziwnego: du mówi, że katalog logów zajmuje dwa gigabajty, a df pokazuje dysk zapełniony w dziewięćdziesięciu ośmiu procentach. Okazuje się, że proces bazy trzyma otwarty plik logu, który administrator usunął wcześniej przez pomyłkę, więc miejsce nie wraca, dopóki proces nie zostanie zrestartowany.

Po co to istnieje

Pojęcie: twardy link i licznik odwołań. W systemie plików Linuksa nazwa pliku widoczna w katalogu (`ls`) i sam plik (jego zawartość, przechowywana w inode'ie) to dwie osobne rzeczy, połączone przez **twardy link** — wpis w katalogu wskazujący na numer inode'u. Ten sam plik może mieć kilka nazw naraz (kilka twardych linków w różnych katalogach) — usunięcie jednej z nich nie kasuje zawartości, tylko usuwa jeden wpis. Jądro liczy, ile takich wpisów wskazuje na dany inode — to jest **licznik odwołań** (`link count`, kolumna `NLINK` w `ls -l` i `lsof`). Dopiero gdy licznik spadnie do zera, jądro wie, że żaden katalog już nie wskazuje na ten plik.

Zwykłe `rm plik.txt` nie „kasuje pliku" w potocznym rozumieniu — technicznie wykonuje operację `unlink()`: usuwa nazwę z katalogu i zmniejsza licznik odwołań o jeden. Jeśli był to jedyny wpis (licznik spadał z 1 do 0) **i** żaden proces nie ma tego pliku otwartego, jądro natychmiast zwalnia zajmowane przez niego bloki na dysku. Ale jeśli w tym samym momencie jakiś proces trzyma ten plik otwarty — ma go w swojej tablicy deskryptorów — jądro nie zwalnia bloków. Plik istnieje dalej, bez nazwy, dostępny wyłącznie przez ten jeden otwarty deskryptor, dopóki proces go nie zamknie albo nie zakończy działania.

Ten mechanizm nie jest usterką — to świadoma decyzja projektowa z pierwszych wersji Uniksa, która pozwala bezpiecznie nadpisywać i usuwać pliki, na których w danym momencie coś pracuje, bez blokowania operacji i bez ryzyka, że proces nagle straci dostęp do pliku w środku odczytu. Konsekwencją jest właśnie ten scenariusz: `df` (który pyta jądro o zajęte bloki) i `du` (który liczy pliki po nazwach w katalogach) zaczynają pokazywać różne liczby.

Analogia

Wyobraź sobie **wypożyczalnię, w której ktoś zwraca klucz do skrytki, ale zapomina, że sam wciąż trzyma w ręku duplikat**.

Recepcja skreśla skrytkę z listy wynajętych — formalnie skrytka jest teraz wolna, nikt jej „nie ma". Gdyby ktoś przyszedł spytać o listę wynajętych skrytek (to jest `du` — patrzy na listę nazw), tej konkretnej już by na niej nie było. Ale klient wciąż fizycznie trzyma klucz i wciąż trzyma w skrytce swoje rzeczy — skrytka jest zajęta naprawdę, mimo że administracyjnie „nie istnieje". Gdyby ktoś policzył realnie zajętą przestrzeń w magazynie (to jest `df` — patrzy na fizycznie zajęte miejsce), ta skrytka wliczyłaby się do sumy, bo rzeczy w niej fizycznie leżą.

Dopiero gdy klient odda duplikat klucza (proces zamyka deskryptor — kończy działanie albo zostaje zrestartowany), obsługa może faktycznie opróżnić skrytkę i oddać ją do puli wolnych. Do tego momentu żadna inwentaryzacja po nazwach (`du`) nie znajdzie tej skrytki, bo formalnie została wykreślona — a mimo to zajmuje realne miejsce w magazynie.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: deskryptor trzymający usunięty plik.** Deskryptor pliku to numer w prywatnej tablicy procesu, wskazujący na wpis w globalnej tablicy otwartych plików jądra, a ten z kolei — na konkretny inode na dysku. Ta ścieżka (deskryptor → wpis jądra → inode) jest niezależna od nazwy pliku w katalogu. Kiedy proces otwiera plik (`open()`), dostaje deskryptor wskazujący bezpośrednio na inode — i ten wskaźnik pozostaje ważny nawet wtedy, gdy plik zniknie ze wszystkich katalogów. Właśnie dlatego `nginx` z przykładu w Fiszce może bez przeszkód dalej dopisywać kolejne linie do swojego loga, mimo że `ls /var/log/nginx/` w ogóle go nie pokazuje — dopisuje przez stary, wciąż ważny deskryptor, a nie przez nazwę.

Najczęstszy sprawca tego scenariusza w praktyce to **logrotate skonfigurowany bez `copytruncate`**. Domyślny mechanizm rotacji przenosi plik (`mv access.log access.log.1`) i tworzy nowy, pusty plik pod starą nazwą, po czym wysyła usłudze sygnał, żeby ta otworzyła nowy uchwyt (np. `nginx -s reopen` albo `SIGHUP`). Jeśli ten sygnał z jakiegoś powodu nie dotrze — usługa nie obsługuje go poprawnie, skrypt logrotate ma błąd, proces akurat jest zawieszony — `nginx` dalej pisze do starego inode'u, który po rotacji i ewentualnym `rm` (przy trybie bez `copytruncate` skrypt czasem i tak usuwa najstarszy plik z rotacji) staje się dokładnie tym „usuniętym, ale otwartym" plikiem. Plik rośnie w nieskończoność, bo proces o tym nie wie, a nazwa `access.log`, którą widzi `du`, to już zupełnie inny, świeży, mały plik.

`df` nie ma pojęcia o nazwach — czyta wyłącznie liczbę zaalokowanych bloków na systemie plików, a te pozostają zajęte, dopóki istnieje choćby jeden otwarty deskryptor **lub** choćby jedna nazwa w katalogu. `du` nie ma pojęcia o deskryptorach — chodzi po drzewie katalogów i sumuje to, co ma nazwę. Rozjazd między nimi to nie błąd żadnego z narzędzi, tylko dokładne odzwierciedlenie dwóch różnych rzeczy, o które pytają.

Krok po kroku

1. Potwierdź rozjazd między `df` a `du`.

```
df -h /var
du -sh /var/* 2>/dev/null | awk '{sum+=$1} END{print sum "G (przyblizone)"}'
```

```
Filesystem                  Size  Used Avail Use% Mounted on
/dev/mapper/vgVAR-lvVAR        30G   29G  380M  99% /var
16G (przyblizone)
```

29G zajęte według `df`, 16G widoczne według `du` — 13G różnicy do wyjaśnienia.

2. Znajdź usunięte, wciąż otwarte pliki na tym systemie plików.

```
sudo lsof -nP +L1 /var
```

```
COMMAND    PID   USER   FD   TYPE DEVICE SIZE/OFF NLINK   NODE NAME
nginx     4821   nginx   6w   REG  253,1  13958643712     0 524301 /var/log/nginx/access.log
(deleted)
```

Rozmiar w kolumnie `SIZE/OFF` (ok. 13 GB) odpowiada dokładnie brakującej różnicy.

3. Zdecyduj: restart czy `truncate` na żywo.

Jeśli przestój usługi jest akceptowalny w tej chwili:

```
sudo systemctl restart nginx
```

Jeśli nie — obetnij deskryptor bez przerywania pracy procesu:

```
sudo truncate -s 0 /proc/4821/fd/6
```

4. Zweryfikuj, że miejsce faktycznie wróciło.

```
df -h /var
```

```
Filesystem                  Size  Used Avail Use% Mounted on
/dev/mapper/vgVAR-lvVAR        30G   16G   13G  56% /var
```

5. Napraw przyczynę, nie tylko skutek.

```
grep -A3 nginx /etc/logrotate.d/nginx
```

```
/var/log/nginx/*.log {
    daily
    rotate 14
    copytruncate
}
```

Dopisanie (albo poprawienie) `copytruncate` w konfiguracji `logrotate` sprawia, że rotacja obcina istniejący plik w miejscu zamiast go przenosić i tworzyć nowy — usługa cały czas pisze do tego samego inode'u pod tą samą nazwą, więc problem usuniętego, ale otwartego pliku przestaje mieć jak wystąpić dla tego loga. Cena: krótkie okno, w którym dwa zapisy mogą się przeplatać, akceptowalne dla większości logów aplikacyjnych.

Częste błędy

Zabicie procesu sygnałem `SIGKILL`, żeby „na pewno" zwolnić miejsce.

```
sudo kill -9 4821
```

Miejsce faktycznie wróci — zamknięcie ostatniego deskryptora zawsze zwalnia inode z zerowym licznikiem dowiązań, niezależnie od tego, jak proces został zamknięty. Problem w tym, że `SIGKILL` nie daje procesowi szansy na domknięcie połączeń, zapisanie buforów ani czyste zakończenie transakcji — dla `nginx` to zwykle nieszkodliwe, ale dla bazy danych albo kolejki wiadomości to

prosta droga do uszkodzonych danych. Prawidłowa kolejność to `systemctl restart` (sygnał `SIGTERM`, czas na czyste zamknięcie, potem dopiero `SIGKILL` jeśli proces nie zareaguje) albo `truncate` na deskryptorze, jeśli restart nie wchodzi w grę.

Obcięcie niewłaściwego deskryptora.

```
sudo truncate -s 0 /proc/4821/fd/3
```

```
(brak błędu — polecenie się wykonuje)
```

`truncate` nie sprawdza za Ciebie, czy deskryptor `3` to ten usunięty plik, czy aktywny, wciąż nazwany plik konfiguracyjny albo otwarte gniazdo sieciowe. Bez wcześniejszego potwierdzenia przez `ls -l /proc/PID/fd`, że dany numer ma dopisek `(deleted)`, ryzykujesz wyzerowanie pliku, który usługa aktywnie czyta lub do którego pisze w regularnym trybie — utrata danych bez żadnego ostrzeżenia w terminalu.

Uruchomienie `lsof` bez `-n`/`-P` na serwerze z wieloma połączeniami.

```
sudo lsof +L1
```

Na serwerze z setkami otwartych gniazd sieciowych `lsof` domyślnie próbuje rozwiązać każdy adres IP na nazwę hosta (DNS) i każdy numer portu na nazwę usługi (`/etc/services`) — polecenie, które powinno zająć ułamek sekundy, wisi kilkanaście sekund albo dłużej, zwłaszcza gdy serwer DNS jest wolny lub nieosiągalny. Naprawa to dodanie `-n -P`, jak w Fiszce, co całkowicie wyłącza te rozwiązania nazw.

Założenie, że `rm` na widocznym pliku od razu zwalnia miejsce.

```
rm /var/log/app/app.log
df -h /var
```

```
Filesystem                  Size  Used Avail Use% Mounted on
/dev/mapper/vgVAR-lvVAR        30G   29G  380M  99% /var
```

Plik znika z `ls`, ale `df` pokazuje dokładnie tyle samo zajętego miejsca co przed chwilą — bo jakiś proces (np. `tail -f` uruchomiony w innej sesji administratora, albo sama aplikacja) wciąż trzyma ten plik otwarty. Miejsce wróci dopiero, gdy ten proces zamknie deskryptor — sprawdź to od razu przez `lsof -nP +L1`, zamiast zakładać usterkę systemu plików.

□□ Ćwiczenia

>

> Wszystko poniżej wykonujesz w `/tmp`, na własnych plikach. `truncate` na deskryptorze wymaga, żeby proces, który go trzyma, należał do Ciebie — jeśli pracujesz na koncie zwykłego użytkownika, cały scenariusz i tak zadziała, bo sam uruchamiasz proces trzymający plik.

Ćwiczenie 1 — odtworzenie klasycznego rozjazdu

```
cd /tmp
dd if=/dev/zero of=demo.log bs=1M count=200 2>/dev/null
tail -f demo.log > /dev/null &
PID=$!
echo "PID procesu trzymającego plik: $PID"

du -sh demo.log
rm demo.log
ls demo.log 2>&1
du -sh /tmp 2>/dev/null | tail -1
```

Plik zniknął z `ls`. Sprawdź trzema sposobami, że wciąż zajmuje miejsce i który proces go trzyma.

```
sudo lsof -nP +L1 | grep tail
ls -l /proc/$PID/fd | grep deleted
```

Odpowiedź

`lsof -nP +L1` (albo `ls -l /proc/$PID/fd`) pokaże wpis `demo.log (deleted)` przy deskryptorze procesu `tail`, z rozmiarem ok. 200 MB w kolumnie `SIZE/OFF` — dokładnie tyle, ile zajmował plik przed `rm`. `du -sh /tmp` w ogóle go nie uwzględni, bo plik nie ma już nazwy w żadnym katalogu. To jest w miniaturze dokładnie ten sam mechanizm, co przy `access.log` z Fiszki, tylko wywołany ręcznie zamiast przez logrotate.

Ćwiczenie 2 — zwolnienie miejsca bez zabijania procesu

Kontynuacja Ćwiczenia 1, z tym samym `$PID`.

```
sudo truncate -s 0 /proc/$PID/fd/1
sudo lsof -nP +L1 | grep tail
kill $PID
```

Sprawdź, czy proces `tail` po `truncate` dalej działa (nie dostał żadnego sygnału) i czy wpis w `lsof` zniknął.

Odpowiedź

Proces `tail` nadal żyje — `truncate` nie wysyła do niego żadnego sygnału, tylko zeruje zawartość pliku, na który wskazuje jego deskryptor. Po obcięciu wpis `(deleted)` znika z wyniku `lsof +L1`, bo licznik dowiązań i tak pozostaje `0`, ale rozmiar pliku (kolumna `SIZE/OFF`) spada do zera — miejsce na dysku zostało zwolnione, mimo że `tail` przez cały czas trzymał (i wciąż trzyma) ten sam, ważny deskryptor. Numer deskryptora może się różnić od `1` w zależności od tego, jak powłoka przekierowała `stdout` procesu w tle — jeśli `ls -l /proc/$PID/fd` z Ćwiczenia 1 pokazał inny numer, użyj go zamiast `1`.

Pytania kontrolne

1. Dlaczego `du -sh` nigdy nie znajdzie pliku, który jest usunięty, ale wciąż otwarty przez proces?

Odpowiedź

`du` chodzi po drzewie katalogów i liczy pliki, które mają nazwę — czyli wpis w jakimś katalogu. Usunięty plik nie ma już żadnej nazwy (licznik dowiązań spadł do zera), więc `du` nie ma jak na niego trafić, mimo że fizycznie wciąż zajmuje bloki na dysku, dopóki trzyma go otwarty choćby jeden proces.

2. Czym różni się `truncate -s 0 /proc/PID/fd/N` od zwykłego usunięcia pliku (`rm`)?

Odpowiedź

`rm` usuwa nazwę pliku z katalogu (zmniejsza licznik dowiązań), ale jeśli proces trzyma plik otwarty, miejsce nie zostaje zwolnione — dokładnie ten problem opisuje ta strona. `truncate` na deskryptorze procesu działa inaczej: nie dotyka nazwy (ta już może nie istnieć), tylko zeruje zawartość inode'u, na który wskazuje aktywny deskryptor — miejsce wraca natychmiast, a proces dalej ma ten sam, ważny uchwyt i może przez niego pisać od nowa, bez restartu.

3. `lsof -nP +L1` nic nie pokazuje, mimo że `df` i `du` się rozjeżdżają o kilka gigabajtów. Co sprawdzisz dalej?

Odpowiedź

`findmnt -R` na podejrzanym katalogu — możliwe, że różnica bierze się nie z usuniętego otwartego pliku, tylko z osobnego systemu plików zamontowanego głębiej w drzewie, którego `du` bez `-x` policzyłoby razem z resztą, ale konkretne polecenie diagnostyczne akurat go pominęło. Warto też sprawdzić rezerwę 5% dla roota (`tune2fs -l`, poprzednia strona) — na dużym wolumenie ext4 to też potrafi dać różnicę rzędu kilku gigabajtów.

4. Dlaczego warto skonfigurować `copytruncate` w `logrotate` dla usług, które nie obsługują sygnału ponownego otwarcia logów?

Odpowiedź

Bez `copytruncate` rotacja przenosi plik pod inną nazwę i tworzy nowy — proces, który nie dostanie albo nie obsłuży sygnału ponownego otwarcia pliku, dalej pisze do starego, przeniesionego (a czasem już usuniętego przez kolejną rotację) inode'u, który rośnie w nieskończoność jako usunięty, ale otwarty plik. `copytruncate` zamiast tego kopiuje bieżącą zawartość, a oryginalny plik obcina w miejscu do zera — usługa cały czas pisze do tego samego inode'u pod tą samą nazwą, więc problem nie ma jak wystąpić.

Powiązane: Co zajmuje miejsce: df, du, lsblk, lsscsi · journalctl i zarządzanie dziennikiem systemowym · systemd: jednostki, stany, zależności · Sygnały: kill, pkill, killall, trap

Źródła: `man 8 lsof` · `man 1 truncate` · `man 5 proc` (sekcja `/proc/[pid]/fd/`) · `man 1 unlink` (POSIX) · `man 8 logrotate`

Revision #16
Created 2026-09-17 21:43:01 UTC by Tomasz Konieczny
Updated 2026-10-03 19:08:42 UTC by Tomasz Konieczny