Zarządzanie procesami

Proces, najogólniej rzecz ujmując, jest wykonywanym programem. Na linuxowy proces składają się:

- **Liniowa przestrzeń adresowa**, w której z kolei można wydzielić sekcję tekstu zawierającą kod programu, sekcję danych, zawierającą dane umieszczone w dynamicznie alokowanej pamięci, oraz sekcję stosu.
- **Licznik programu** wskazujący na aktualnie wykonywaną instrukcję, lub wiele takich liczników w przypadku programów wielowątkowych.
- **Zapis stanu rejestrów** procesora.
- **Deskryptory plików**, opisujące otwarte przez proces pliki, strumienie i połączenia sieciowe.
- **Dane procesu**, takie jak unikalny dla procesu numer PID, numer użytkownika UID, numery grup GID, efektywny UID i GID, czy też liczba nice.
- **Zależności rodzinne** z innymi procesami: rodzic, lista dzieci i braci.
- **Liczniki statystyczne** zapisujące m.in. zużyty czas w trybie użytkownika i trybie jądra.

Każdy proces w Linuxie znajduje się w jednym ze stanów:

- **Pracujący w trybie użytkownika** - proces znajduje się na procesorze i wykonuje swój kod.
- **Pracujący w trybie jądra** - jądro wykonuje wywołanie systemowe wykonane przez proces.
- **Uśpiony** - proces czeka na jakieś zdarzenie, na przykład na odczyt danych z dysku lub otrzymanie danych z sieci.
- **Gotowy do wykonania** - może być uruchomiony w każdej chwili, jednak nie ma jeszcze przydzielonego procesora.
- **Zombie** - proces zakończył działanie i czeka na odebranie kodu powrotu przez proces macierzysty.

Procesy w Linuxie mogą powstać tylko w jeden sposób - przez duplikację istniejącego procesu podczas wywołania systemowego. Jedynym wyjątkiem jest proces init o numerze 1, który tworzony jest od zera przez kernel podczas uruchamiania systemu. Uruchomienie innego programu przez proces następuje przez wywołanie execve.

Każdy proces, z racji sposobu powstawania, posiada swój proces macierzysty. Proces taki posiada zarówno swoje prawa, jak i obowiązki. Proces macierzysty może nadzorować pracę procesu potomnego. Do obowiązków należy odczytanie kodu powrotnego procesu. Jeśli kod powrotny nie zostanie odczytany, zakończony proces będzie wciąż widoczny na liście procesów jako zombie.

Procesy linuksowe poddawane są szeregowaniu z wywłaszczaniem. Każdy proces, który zostaje umieszczony na procesorze, jest utrzymywany na nim tylko przez określony kwant czasu, a następnie usuwany, żeby na jego miejsce mógł wejść inny. Stosowany jest system priorytetów, pozwalający tak ustawić intensywnie używające procesor procesy tła, aby nie blokowały pracy procesom interakcyjnym. Jednak, niezależnie od priorytetu, każdy proces musi otrzymać kwant czasu; nie ma możliwości dławienia przez procesy wysokopriorytetowe.

Obok proces uruchamianych przez użytkowników, istnieją również tak zwane demony (z angielskiego daemon). Są to procesy uruchamiane przy starcie systemu, odpowiedzialne za realizacje niektórych funkcji serwera. Przykładami demonów mogą być: inetd (demon Internetowy, odpowiedzialny za uruchamianie program usługowych po stronie serwera, takich jak telnet, ftp itp), syslogd i klogd (odpowiedzialne za logi systemowe), named (serwer DNS), httpd (serwer WWW) itd. Cechą charakterystyczną dla demonów jest nazwa kończąca się literą "d". Demony można porównać do program rezydentnych (TSR) w systemie MS-DOS.

Informacje o procesach - ps

W systemie Linux informacje o pracujących procesach udostępnione są w podkatalogach /proc o nazwie takiej, jak PID procesu. Z informacji tych korzysta program ps, który pozwala na wyświetlenie ich w uporządkowanej, tabelarycznej postaci. Program ps przyjmuje liczne parametry, pozwalając dopasować ilość wyświetlanych procesów i wyświetlane informacje do chwilowych potrzeb. Wykonajmy ps bez żadnych argumentów:

```
tilk@lucifer:~$ ps
  PID TTY          TIME CMD
21592 6pts/16   00:00:00 bash
22263 pts/16    00:00:00 ps
tilk@lucifer:~$
```

Wyświetlany jest numer PID, terminal sterujący procesu, całkowity czas, w którym proces zajmował procesor, oraz komenda, za pomocą której proces został uruchomiony. Wyświetlono jedynie te procesy, które pracują na tym samym terminalu, co użytkownik. Aby wyświetlić procesy, które są na innych terminalach lub nie posiadają żadnego terminala, należy użyć opcji x:

```
tilk@lucifer:~$ ps x
  PID TTY      STAT   TIME COMMAND
20673 pts/13   S      0:00 bash
32152 pts/13   S      0:00 mc
(...)
tilk@lucifer:~$
```

Procesy innych użytkowników, które posiadają terminale sterujące, zobaczymy, używając opcji a:

```
tilk@lucifer:~$ ps a
  PID TTY      STAT   TIME COMMAND
15324 tty4     S      0:00 /sbin/getty 38400 tty4
16310 tty5     S      0:00 /sbin/getty 38400 tty5
12040 tty6     S      0:00 /sbin/getty 38400 tty6
(...)
tilk@lucifer:~$
```

Wszystkie procesy pracujące w systemie uzyskamy, łącząc ze sobą opcje a oraz x:

```
tilk@lucifer:~$ ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:04 init [2]
    2 ?        SW     0:19 [keventd]
    3 ?        SW     0:00 [kapmd]
    4 ?        SWN    0:14 [ksoftirqd_CPU0]
    5 ?        SW     2:16 [kswapd]
    6 ?        SW     0:00 [bdflush]
    7 ?        SW     0:04 [kupdated]
   11 ?        SW     2:58 [kjournald]
(...)
tilk@lucifer:~$
```

Oprócz opcji filtrujących są też opcje, które zmieniają rodzaj wyświetlanych danych. Zdecydowanie najprzydatniejszą z nich jest opcja u, która powoduje włączenie formatu użytkowników:

```
tilk@lucifer:~$ ps u
USER       PID %CPU %MEM    VSZ  RSS TTY      STAT START   TIME COMMAND
tilk     18469  0.0  0.4   4788 1760 tty1     S    Feb24   0:00 -bash
tilk      4491  0.0  0.4   4656 1764 tty2     S    Feb24   0:00 -bash
tilk      8791  0.0  0.4   4656 1836 tty3     S    Feb24   0:00 -bash
(...)
tilk@lucifer:~$
```

Jak widać, pole COMMAND znacznie się zwęziło, ustępując miejsca ważniejszym danym. Widać również, że wyświetlane są procesy z innych terminali - opcja u implikuje więc x.

Inną przydatną opcją formatującą jest opcja f. Powoduje ona wyświetlenie drzewa procesów, uwzględniając zależność proces macierzysty - proces potomny:

```
tilk@lucifer:~$ ps f
  PID TTY      STAT   TIME COMMAND
19376 pts/17   S      0:00 bash
30005 pts/17   R      0:00  \_ ps f
20673 pts/13   S      0:00 bash
32152 pts/13   S      0:00  \_ mc
15158 pts/15   S      0:00      \_ bash -rcfile .bashrc
(...)
tilk@lucifer:~$
```

Pozostałymi opcjami formatującymi są: l dla formatu długiego, j dla formatu prac, s dla formatu sygnałów oraz v dla formatu pamięci wirtualnej. Informacje o nich, jak również opisy znaczeń poszczególnych pól, znajdują się w podręczniku systemowym.

Statystyki procesów - top

Top jest programem działającym w czasie rzeczywistym, prezentującym najbardziej pożerające procesor i pamięć procesy w systemie. Po uruchomieniu, ekran terminala wygląda następująco:

```
 20:50:46 up 21 days,  5:22, 35 users,  load average: 0,66, 0,54, 0,43
 242 processes: 239 sleeping, 2 running, 1 zombie, 0 stopped
 CPU states:   9,3% user,  18,4% system,   0,1% nice,  72,2% idle
 Mem:    386248K total,   369808K used,    16440K free,    34032K buffers
 Swap:   457844K total,   170436K used,   287408K free,   112924K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 9248 tilk      19   0  1916 1916  1584 R    12,4  0.4   0:00 top
    1 root       8   0   808  772   752 S     0,0  0.1   0:04 init
    2 root       9   0     0    0     0 SW    0,0  0.0   0:19 keventd
    3 root       9   0     0    0     0 SW    0,0  0.0   0:00 kapmd
(...)
```

Standardowo, procesy sortowane są według zużycia procesora. Można jednak przełączyć sortowanie, naciskając jeden z klawiszy:

- N - według numeru PID
- A - według wieku
- P - według użycia procesora
- M - według użycia pamięci
- T - według czasu pracy

Lista jest automatycznie odświeżana, można jednak wymusić odświeżanie, wciskając klawisz spacji. Częstotliwość odświeżania możliwa jest do ustawienia za pomocą klawisza s.

Możliwe jest również dopasowanie wyświetlanych kolumn. Jest to rzadko wykorzystywana możliwość; detale znajdują się w podręczniku systemowym.

Linuxowe sygnały. Kill

Jedną z typowo linuksowych metod komunikacji proces - proces i jądro - proces są sygnały. Sygnały są mechanizmem asynchronicznym - proces po otrzymaniu sygnału przerywa pracę i wykonuje kod obsługi sygnału. Część sygnałów służy do komunikowania procesu o kluczowych wydarzeniach przez jądro, takie jak słynny SIGSEGV, sygnalizujący błąd dostępu do pamięci. Te sygnały nie mogą być ignorowane. Inne sygnały przeznaczone są do użycia w komunikacji międzyprocesowej. Oto tabela najczęściej używanych sygnałów:

nazwa	numer	dom. akcja	opis
SIGHUP	1	zakończenie	Wyłączenie terminala sterującego bądź śmierć procesu kontrolującego
SIGINT	2	zakończenie	Przerwanie z klawiatury (CTRL+C)
SIGQUIT	3	zrzut core	Wyjście nakazane z klawiatury
SIGILL	4	zrzut core	Próba wykonania nieprawidłowej instrukcji
SIGABRT	6	zrzut core	Sygnał przerwania pracy procesu wywołany przez abort()
SIGKILL	9	zakończenie	Natychmiastowe usunięcie procesu; niemożliwy do złapania ani zignorowania.
SIGSEGV	11	zrzut core	Nieprawidłowe odwołanie do pamięci wirtualnej
SIGPIPE	13	zakończenie	Zerwany potok: pisanie do potoku, który nie posiada procesu po stronie czytania
SIGALRM	14	zakończenie	Sygnał alarmowy wywołany przez funkcję alarm()
SIGTERM	15	zakończenie	Sygnał zakończenia pracy procesu
SIGCHLD	17	ignorowanie	Zatrzymanie bądź wyłączenie procesu potomnego
SIGCONT	18	start	Kontynuacja zatrzymanego procesu
SIGSTOP	19	zatrzymanie	Zatrzymanie procesu; niemożliwy do złapania ani ignorowania

Informacja o pozostałych, rzadko używanych sygnałach znajduje się w podręczniku systemowym pod stroną signal(7).

Możliwe jest ręczne wysłanie sygnału do procesu. Do tego celu wykorzystuje się polecenie kill. Domyślnie wysyła ono sygnał SIGTERM, może jednak wysłać dowolny sygnał. Spróbujmy teraz stworzyć zadanie w tle, a następnie je unicestwić:

```
tilk@lucifer:~$ cat /dev/zero > /dev/null &
[1] 25654
tilk@lucifer:~$ kill 25654
```

```
tilk@lucifer:~$
[1]+  Zakończony              cat /dev/zero >/dev/null
tilk@lucifer:~$
```

Na szczególnie uparte programy można użyć sygnału SIGKILL:

```
tilk@lucifer:~$ cat /dev/zero > /dev/null &
[1] 8606
tilk@lucifer:~$ kill -9 8606
tilk@lucifer:~$
[1]+  Unicestwiony            cat /dev/zero >/dev/null
tilk@lucifer:~$
```

Do wysyłania sygnałów do procesów określonych nie numerem PID, lecz nazwą komendy, służy polecenie killall o takiej samej składni, jak kill, lecz na miejscu numeru PID przyjmujące nazwę.

uruchamianie procesów w tle i ich przywracanie

Procesy uruchomione w tle to takie, które nie blokują nam dostępu do powłoki np. włączamy muzykę i korzystamy z edytora tekstowego. Aby uruchomić proces w tle, należy na końcu polecenia dodać znak & np.
vi tekscik & - spowoduje to przeniesienie programu **vi** do tła, gdzie będzie czekał na przywrócenie przez użytkownika (tekscik to plik tekstowy). Aby je przywrócić na konsole, należy skorzystać z polecenia jobs jako wynik otrzymamy ponumerowaną listę naszych zadań np.

```
[1] - Suspended (tty output) vi
[2] + Suspended (tty input) cat >> ak
```

w nawiasach kwadratowych mamy podany numer zadania, który przekazujemy jako argument do polecenia fg, np.
fg 2 spowoduje przywrócenie zadania **cat** na terminal, jednak jeśli wydamy samo polecenie fg również przywrócone zostanie zadanie cat.
Uruchomione już procesy przenosimy do tła za pomocą klawiszy CTRL+Z (^Z) gdzie zostaną one zatrzymane. Aby ponownie je uruchomić ale już w tle używamy polecenia bg z numerem zadania.
np.
sleep - jest to polecenie blokujące dostęp do terminala na określony argumentem czas w sekundach
np. sleep 100, po uruchomieniu CTRL+Z => Suspended - informacja, że program został zatrzymany

```
s109719@geolog:~> jobs
[1] Suspended (tty output) vi
[2] - Suspended (tty output) top
[3] + Suspended sleep 100
```

pod numerem 3 nasz program czeka w tle

```
s109719@geolog:~> bg 3
[3] sleep 100 & - informacja o przeniesieniu programu sleep do tła (uruchomionego)

s109719@geolog:~> jobs
```

[1] - Suspended (tty output) vi
[2] + Suspended (tty output) top
[3] Running sleep 100
program działa sobie w tle s109719@geolog:~>
[3] Done sleep 100
po 100 sekundach program zakończył działanie, co zasygnalizował przez słowo Done
Aby zatrzymać proces uruchomiony w tle należy wydać polecenie stop PID
np.
proces sleep działa w tle:
s109719@geolog:~>jobs [1] Suspended (signal) vi
[2] - Suspended (tty output) top
[3] Running sleep 1000 - proces sleep działa

s109719@geolog:~> ps - otrzymamy PID procesu

PID TTY TIME CMD
19977 pts/2 00:00:00 tcsh
22929 pts/2 00:00:00 vi
30827 pts/2 00:00:00 top
30854 pts/2 00:00:00 sleep
wykonujemy polecenie stop 30854

[3] + Suspended (signal) sleep 1000 - proces został zatrzymany

s109719@geolog:~> jobs - dla potwierdzenia, sprawdzamy status procesu poleceniem jobs
[1] Suspended (signal) vi
[2] Suspended (tty output) top
[3] + Suspended (signal) sleep 1000