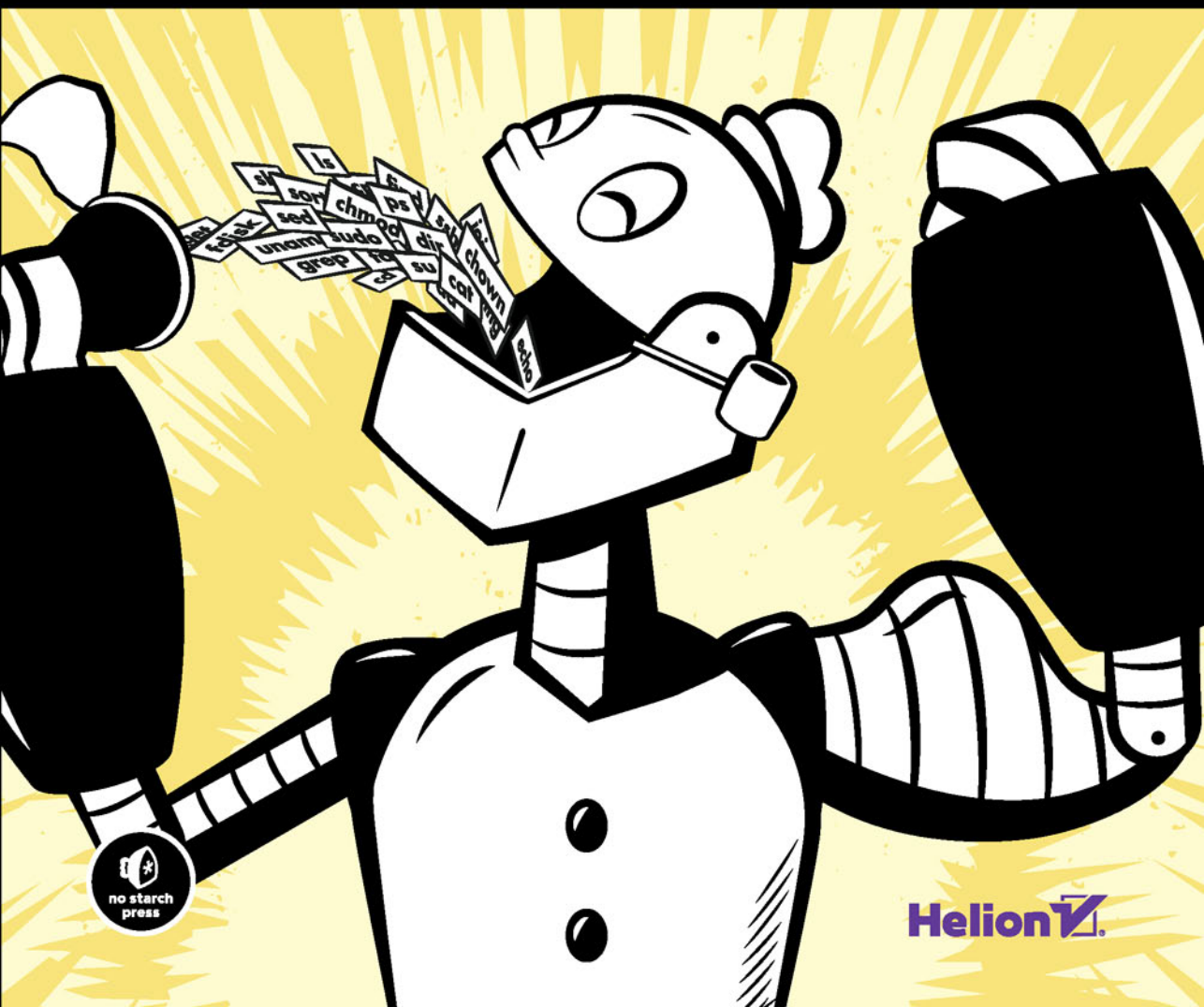
LINUX

WPROWADZENIE DO WIERSZA POLECEŃ

WILLIAM E. SHOTTS JR



Helion

Tytuł oryginału: The Linux Command Line

Tłumaczenie: Joanna Zatorska (wstęp, rozdz. 3 – 36),
Przemysław Szeremiota (rozdz. 1 – 2)

ISBN: 978-83-283-0174-0

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Spis treści

Część II
Konfiguracja i środowisko

Część III
Popularne zadania i podstawowe narzędzia

Część IV
Tworzenie skryptów powłoki

10

Procesy

Współczesne systemy operacyjne są zwykle *wielozadaniowe*, co oznacza, że poprzez szybką zmianę programu, który jest wykonywany, sprawiają wrażenie wykonywania kilku zadań jednocześnie. Jądro systemu Linux zarządza tym poprzez wykorzystanie *procesów*. Procesy są sposobem organizowania przez Linux różnych programów oczekujących na swoją kolej do zasobów CPU.

Niekiedy komputer staje się powolny lub aplikacja przestaje odpowiadać. W tym rozdziale przyjrzymy się niektórym narzędziom dostępnym w wierszu poleceń, które umożliwiają nam sprawdzenie, co wykonują programy i jak zakończyć procesy, które przebiegają niewłaściwie.

W tym rozdziale zostaną wprowadzone następujące polecenia:

- `ps` — wyświetla listę bieżących procesów,
- `top` — wyświetla zadania,
- `jobs` — wypisuje aktywne zadania,
- `bg` — umieszcza zadanie w tle,
- `fg` — umieszcza zadanie na pierwszym planie,
- `kill` — wysyła sygnał do procesu,
- `killall` — kończy proces o podanej nazwie,
- `shutdown` — zamyka lub ponownie uruchamia system.

Jak działa proces?

Podczas rozruchu systemu jądro inicjalizuje kilka własnych zadań w postaci procesów i uruchamia program zwany `init`. Z kolei `init` uruchamia serię skryptów powłoki (znajdujących się w katalogu */etc*) zwanych *skryptami inicjalizacyjnymi*, które uruchamiają wszystkie usługi systemowe. Wiele z tych usług jest zaimplementowanych w postaci demonów, czyli programów, które wykonują swoje zadania w tle i nie posiadają żadnego interfejsu użytkownika. Dlatego nawet jeśli nie jesteśmy zalogowani, system jest zawsze choć trochę zajęty wykonywaniem rutynowych zadań.

Możliwość uruchamiania programów przez inne programy jest wyrażana na drzewie procesów w postaci *procesu macierzystego* tworzącego *proces potomny*.

Jądro ułatwia sobie organizację, utrzymując informacje o każdym procesie. Na przykład do każdego procesu przypisywany jest numer zwany *identyfikatorem procesu* (*PID* — ang. *process id*). Numery PID są przydzielane w kolejności rosnącej, przy czym `init` zawsze otrzymuje numer 1. Jądro kontroluje także pamięć przydzieloną do każdego procesu, a także gotowość procesów do wznowienia działania. Podobnie jak pliki, procesy również posiadają właścicieli, identyfikatory użytkownika, identyfikatory EUID itd.

Wyświetlanie procesów za pomocą polecenia ps

Najczęściej używanym poleceniem służącym do wyświetlania procesów (jest ich kilka) jest `ps`. Program `ps` przyjmuje wiele opcji, jednak w najprostszy sposób używa się go następująco:

```
[me@linuxbox ~]$ ps
  PID TTY          TIME CMD
 5198 pts/1    00:00:00 bash
10129 pts/1    00:00:00 ps
```

W wyniku działania powyższego polecenia uzyskaliśmy listę dwóch procesów: 5198 i 10129, które odpowiadają programom `bash` i `ps`. Jak widać, domyślnie polecenie `ps` nie wyświetla zbyt wiele, tylko procesy związane z bieżącą sesją terminala. Aby zobaczyć więcej, musimy dodać kilka opcji. Zanim to zrobimy, spójrzmy na inne pola wyświetlone przez `ps`. `TTY` jest skrótem od słowa *teletype* i odnosi się do *terminala kontrolnego* procesu. Jest to pozostałość po dawnych czasach systemu Unix. Pole `TIME` oznacza czas wykorzystania CPU przez proces. Jak widać, żaden z procesów nie wymagał od komputera dużego nakładu pracy.

Po dodaniu opcji możemy uzyskać lepszy obraz działania systemu:

```
[me@linuxbox ~]$ ps x
  PID TTY      STAT   TIME COMMAND
 2799 ?        Ssl    0:00 /usr/libexec/bonobo-activation-server –ac
 2820 ?        Sl     0:01 /usr/libexec/evolution-data-server-1.10 --
15647 ?        Ss     0:00 /bin/sh /usr/bin/startkde
```

```
15751 ?        Ss     0:00 /usr/bin/ssh-agent /usr/bin/dbus-launch --
15754 ?        S      0:00 /usr/bin/dbus-launch --exit-with-session
15755 ?        Ss     0:01 /bin/dbus-daemon --fork --print-pid 4 –pr
15774 ?        Ss     0:02 /usr/bin/gpg-agent -s –daemon
15793 ?        S      0:00 start_kdeinit --new-startup +kcminit_start
15794 ?        Ss     0:00 kdeinit Running...
15797 ?        S      0:00 dcopserver –nosid
i wiele więcej...
```

Dodanie opcji `x` (zauważmy brak myślnika przed opcją), informuje `ps` o konieczności wyświetlenia informacji o wszystkich procesach, niezależnie od terminala, pod którego kontrolą pozostają (o ile taki istnieje). Obecność znaku `?` w kolumnie `TTY` oznacza brak terminala kontrolnego. Korzystając z tej opcji, możemy się dowiedzieć, które procesy należą do nas.

Ponieważ w systemie działa wiele procesów, `ps` wyświetla długą listę. Zwykle warto przekazać potokiem wynik polecenia do polecenia `less`, aby ułatwić przeglądanie. Niektóre kombinacje opcji tworzą też długie wiersze tekstu, dlatego dobrym pomysłem będzie zmaksymalizowanie okna emulatora terminala.

Na liście wynikowej pojawiła się nowa kolumna `STAT`. Nazwa ta jest skrótem od angielskiego wyrazu *state* („stan") i zawiera bieżący stan procesu, zgodnie z opisem w tabeli 10.1.

Tabela 10.1. Stany procesu

Status	Znaczenie
R	Uruchomiony. Proces jest uruchomiony lub gotowy do uruchomienia.
S	Uśpiony. Proces nie działa; raczej oczekuje na zdarzenie, takie jak naciśnięcie klawisza, lub na otrzymanie pakietu sieciowego.
D	Uśpiony nieprzerywalny. Proces oczekuje na urządzenie wejścia-wyjścia, takie jak dysk.
T	Zatrzymany. Proces, który otrzymał instrukcję zatrzymania (więcej informacji na ten temat znajdziemy w dalszej części rozdziału).
Z	Proces działający nieprawidłowo lub proces „zombie". Jest to proces potomny, który zakończył działanie, jednak nie został usunięty przez proces macierzysty.
<	Proces o wysokim priorytecie. Procesowi możemy nadać większe znaczenie, przydzielając mu więcej czasu na CPU. Ta właściwość procesu nosi nazwę *niceness*. Możemy powiedzieć, że proces o wysokim priorytecie jest gorszy, ponieważ potrzebuje więcej czasu CPU, co pozostawia mniej czasu dla pozostałych.
N	Proces o niskim priorytecie. Proces taki (lepszy proces) uzyska dostęp do procesora tylko wtedy, gdy zostaną obsłużone pozostałe procesy, mające wyższy priorytet.

Po znaku stanu procesu mogą wystąpić inne znaki. Oznaczają one różne egzotyczne cechy procesu. Szczegółowe informacje na ten temat można uzyskać w podręczniku man polecenia `ps`.

Inny popularny zestaw opcji to `aux` (bez myślnika przed opcją). Pozwala uzyskać jeszcze więcej informacji:

```
[me@linuxbox ~]$ ps aux
USER     PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       1  0.0  0.0   2136   644 ?        Ss   Mar05   0:31 init
root       2  0.0  0.0      0     0 ?        S<   Mar05   0:00 [kt]
root       3  0.0  0.0      0     0 ?        S<   Mar05   0:00 [mi]
root       4  0.0  0.0      0     0 ?        S<   Mar05   0:00 [ks]
root       5  0.0  0.0      0     0 ?        S<   Mar05   0:06 [wa]
root       6  0.0  0.0      0     0 ?        S<   Mar05   0:36 [ev]
root       7  0.0  0.0      0     0 ?        S<   Mar05   0:00 [kh]
i wiele więcej...
```

Ten zestaw opcji wyświetla procesy należące do każdego użytkownika. Użycie tych opcji bez myślnika sprawia, że polecenie będzie działać w „stylu BDS". Wersja programu `ps` dostępna w systemie Linux może emulować zachowania programu `ps` z kilku dystrybucji Uniksa. Opcje te wyświetlają dodatkowe kolumny, opisane w tabeli 10.2.

Tabela 10.2.

Nagłówek	Znaczenie
USER	Identyfikator użytkownika. Jest to właściciel procesu.
%CPU	Wykorzystanie CPU w procentach.
%MEM	Wykorzystanie pamięci w procentach.
VSZ	Rozmiar pamięci wirtualnej.
RSS	Skrót od *Resident Set Size*. Ilość pamięci fizycznej (RAM) wykorzystywanej przez proces w kilobajtach.
START	Czas rozpoczęcia procesu. Wartości powyżej 24 godzin reprezentowane są przez daty.

Dynamiczne wyświetlanie procesów za pomocą polecenia top

Chociaż polecenie `ps` umożliwia wyświetlenie wielu informacji o działaniu maszyny, to pozwala uzyskać jedynie obraz stanu maszyny w momencie wykonania polecenia `ps`. Aby wyświetlić bardziej dynamiczny obraz aktywności maszyny, korzystamy z polecenia `top`:

```
[me@linuxbox ~]$ top
```

Program `top` wyświetla ciągle uaktualniany (domyślnie co 3 sekundy) obraz procesów w systemie, posortowanych według aktywności. Nazwa polecenia związana jest ze sposobem działania. Otóż program `top` wyświetla tylko procesy z górnej (ang. *top*) części listy procesów. Wynik działania polecenia `top` składa się z dwóch części: podsumowania stanu systemu w górnej części oraz tabeli procesów posortowanych według aktywności:

```
top - 14:59:20 up  6:30,  2 users,  load average: 0.07, 0.02, 0.00
Tasks: 109 total,   1 running, 106 sleeping,   0 stopped,   2 zombie
Cpu(s):  0.7%us,  1.0%sy,  0.0%ni, 98.3%id,  0.0%wa,  0.0%hi,  0.0%si
Mem:    319496k total,   314860k used,     4636k free,    19392k buff
Swap:   875500k total,   149128k used,   726372k free,   114676k cach
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6244 me        39  19 31752 3124 2188 S  6.3  1.0  16:24.42 trackerd
11071 me        20   0  2304 1092  840 R  1.3  0.3   0:00.14 top
 6180 me        20   0  2700 1100  772 S  0.7  0.3   0:03.66 dbus-dae
 6321 me        20   0 20944 7248 6560 S  0.7  2.3   2:51.38 multiloa
 4955 root      20   0  104m 9668 5776 S  0.3  3.0   2:19.39 Xorg
    1 root      20   0  2976  528  476 S  0.0  0.2   0:03.14 init
    2 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migratio
    4 root      15  -5     0    0    0 S  0.0  0.0   0:00.72 ksoftirq
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.04 watchdog
    6 root      15  -5     0    0    0 S  0.0  0.0   0:00.42 events/0
    7 root      15  -5     0    0    0 S  0.0  0.0   0:00.06 khelper
   41 root      15  -5     0    0    0 S  0.0  0.0   0:01.08 kblockd/
   67 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kseriod
  114 root      20   0     0    0    0 S  0.0  0.0   0:01.62 pdflush
  116 root      15  -5     0    0    0 S  0.0  0.0   0:02.44 kswapd0
```

Podsumowanie stanu systemu zawiera wiele przydatnych informacji, które zostały wyjaśnione w tabeli 10.3.

Program `top` przyjmuje zestaw poleceń z klawiatury. Dwa najciekawsze to `h`, które wyświetla ekran pomocy programu, oraz `q`, które zamyka program `top`.

Obydwa największe środowiska graficzne udostępniają aplikacje graficzne, które wyświetlają informacje podobne do `top` (w większości przypadków w podobny sposób, jak czyni to Windows), jednak uważam, że polecenie `top` jest lepsze niż wersje graficzne ze względu na szybkość i znacznie mniejsze wymagania co do zasobów systemu. W końcu nasz program monitorujący system nie powinien przyczyniać się do dalszego spowolnienia systemu, które przecież chcemy wyśledzić.

Sterowanie procesami

Wiemy już, jak wyświetlić informacje o procesach i jak je monitorować. Zobaczmy teraz, jak uzyskać nad nimi pewną kontrolę. Na potrzeby naszych ćwiczeń wykorzystamy mały program o nazwie `xlogo`. Program `xlogo` jest przykładowym programem dostarczanym z systemem X Window System (silnik odpowiadający za wyświetlanie grafiki na monitorze), który po prostu wyświetla okno o zmiennej wielkości zawierające logo *X*. Najpierw poznajmy obiekt naszych eksperymentów:

```
[me@linuxbox ~]$ xlogo
```

Tabela 10.3. Pola informacyjne polecenia top

Rząd	Pole	Znaczenie
1	`top`	Nazwa programu.
	`14:59:20`	Bieżący czas.
	`up 6:30`	Wartość ta zwana jest *uptime*. Jest to czas, który upłynął od ostatniego uruchomienia systemu na maszynie. W tym przykładzie system został uruchomiony 6 i pół godziny temu.
	`2 users`	Zalogowanych jest dwóch użytkowników.
	`load average:`	Wartość *load average* oznacza liczbę procesów oczekujących na wznowienie; czyli jest to liczba procesów, które są uruchomione i dzielą między sobą czas CPU. Wyświetlone są trzy wartości, z których każda dotyczy różnego okresu. Pierwszą jest średnia dla ostatnich 60 sekund, kolejna dotyczy ostatnich 5 minut, a ostatnia dotyczy ostatnich 15 minut. Wartości poniżej 1,0 oznaczają, że maszyna nie jest zbyt zajęta.
2	`Tasks:`	Podsumowanie liczby procesów oraz ich różnych stanów.
	`0.7%us`	Procesy użytkownika wykorzystują 0,7% CPU. Dotyczy to procesów wykonywanych poza samym jądrem.
	`1.0%sy`	Procesy systemu (jądra) wykorzystują 1,0% CPU.
	`0.0%ni`	Procesy o niskim priorytecie wykorzystują 0,0% CPU.
	`98.3%id`	98,3% CPU jest niewykorzystane.
	`0.0%wa`	0,0% CPU oczekuje na urządzenie I/O.
4	`Mem:`	Pokazuje wykorzystanie fizycznej pamięci RAM.
5	`Swap:`	Pokazuje wykorzystanie przestrzeni wymiany (pamięci wirtualnej).

Gdy wpiszemy polecenie, gdzieś na ekranie pojawi się małe okno zawierające logo. W niektórych systemach `xlogo` może wyświetlić ostrzeżenie, które jednak możemy zignorować.

UWAGA *Jeśli w systemie nie zainstalowano programu `xlogo`, można zamiast niego wykorzystać program `gedit` lub `kwrite`.*

Zmieniając rozmiar okna, możemy sprawdzić, czy program `xlogo` pozostaje uruchomiony. Jeśli wielkość logo dopasuje się do nowego rozmiaru, oznacza to, że program działa.

Czy Czytelnik zauważył, że polecenie nie zwróciło wyniku w powłoce? Przyczyną tego zachowania jest fakt, że powłoka oczekuje na zakończenie działania programu, podobnie jak w przypadku pozostałych programów, z których dotychczas korzystaliśmy. Znak zachęty powróci do powłoki, gdy zamkniemy okno `xlogo`.

Zatrzymywanie procesu

Zobaczmy, co się stanie, gdy ponownie uruchomimy `xlogo`. Najpierw wpiszmy polecenie `xlogo` i sprawdźmy, czy program działa. Następnie powróćmy do okna terminala i użyjmy kombinacji klawiszy *Ctrl+C*.

```
[me@linuxbox ~]$ xlogo
[me@linuxbox ~]$
```

Zastosowanie kombinacji klawiszy *Ctrl+C* zatrzymuje program. Oznacza to, że grzecznie prosimy program o zakończenie działania. Po użyciu kombinacji klawiszy *Ctrl+C* okno `xlogo` zostało zamknięte, a do powłoki powrócił znak zachęty.

W ten sposób możemy zatrzymać wiele programów (lecz nie wszystkie).

Umieszczanie procesu w tle

Załóżmy, że chcemy, aby znak zachęty powrócił do powłoki, ale bez zatrzymywania programu `xlogo`. Wykonamy to, umieszczając proces *w tle*. Załóżmy, że terminal ma pierwszy plan (w którym widoczne są różne rzeczy, na przykład znak zachęty) oraz drugi plan, czyli tło (zawierający rzeczy ukryte pod powierzchnią). Aby uruchomić program i natychmiast umieścić go w tle, po poleceniu wpisujemy znak ampersand (&):

```
[me@linuxbox ~]$ xlogo &
[1] 28236
[me@linuxbox ~]$
```

Gdy wpisaliśmy polecenie, na ekranie pojawiło się okno `xlogo`, a do powłoki powrócił znak zachęty, jednak pojawiły się także pewne śmieszne cyferki. Ta informacja wchodzi w skład funkcji powłoki zwanej *sterowaniem zadaniami*. Powłoka informuje nas, że uruchomiła zadanie o numerze 1 (`[1]`) oraz że posiada ono PID 28236. Sprawdźmy nasz proces, wykonując polecenie `ps`:

```
[me@linuxbox ~]$ ps
  PID TTY          TIME CMD
10603 pts/1    00:00:00 bash
28236 pts/1    00:00:00 xlogo
28239 pts/1    00:00:00 ps
```

Funkcja sterowania zadaniami powłoki umożliwia nam również wyświetlenie zadań uruchomionych z poziomu naszego terminala. Wykonując polecenie `jobs`, uzyskamy następującą listę:

```
[me@linuxbox ~]$ jobs
[1]+  Running                 xlogo &
```

Powyższy wynik informuje, że uruchomione jest jedno zadanie o numerze 1 oraz że polecenie miało postać `xlogo &`.

Przywracanie procesu do pierwszego planu

Proces działający w tle nie reaguje na wejście z klawiatury, ignoruje też próby przerwania procesu kombinacją klawiszy *Ctrl+C*. Aby przywrócić proces do pierwszego planu, korzystamy z polecenia `fg`, jak w poniższym przykładzie:

```
[me@linuxbox ~]$ jobs
[1]+  Running                 xlogo &
[me@linuxbox ~]$ fg %1
xlogo
```

Polecenie `fg`, po którym następuje znak procenta oraz numer zadania (zwany *jobspec*), załatwia sprawę. Jeśli w tle uruchomione jest tylko jedno polecenie, numer zadania jest opcjonalny. Aby zakończyć działanie programu `xlogo`, użyjmy kombinacji klawiszy *Ctrl+C*.

Zatrzymywanie (wstrzymywanie) procesu

Czasem chcemy zatrzymać działanie procesu bez jego zakończenia. Możliwość tę wykorzystuje się zwykle, aby można było przenieść proces pierwszoplanowy na drugi plan. Aby zatrzymać proces pierwszoplanowy, należy zastosować kombinację klawiszy *Ctrl+Z*. Spróbujmy. Wpiszmy `xlogo` w wierszu poleceń, naciśnijmy *Enter*, a następnie kombinację klawiszy *Ctrl+Z*.

```
[me@linuxbox ~]$ xlogo
[1]+  Stopped                 xlogo
[me@linuxbox ~]$
```

Po zatrzymaniu programu `xlogo` możemy zweryfikować, czy został zatrzymany, próbując zmienić rozmiar okna programu. Zauważymy, że wydaje się raczej martwe. Program możemy przywrócić na pierwszy plan, korzystając z polecenia `fg`, lub umieścić go w tle, używając polecenia `bg`:

```
[me@linuxbox ~]$ bg %1
[1]+ xlogo &
[me@linuxbox ~]$
```

Jeśli chodzi o polecenie `fg`, to jeśli istnieje tylko jedno zadanie, numer zadania jest opcjonalny.

Umieszczanie procesu pierwszoplanowego w tle jest przydatne, jeśli uruchamiamy program graficzny z wiersza poleceń, ale zapomnimy umieścić go w tle za pomocą znaku & wpisanego za poleceniem.

Dlaczego uruchamia się programy graficzne z wiersza poleceń? Istnieją dwa powody. Pierwszy wynika z tego, że program, który chcemy uruchomić, może nie być wymieniony w menu menedżera okna (na przykład `xlogo`).

Drugi wynika z tego, że program uruchomiony z wiersza poleceń będzie wyświetlać informacje o błędach, które mogłyby być niewidoczne w przypadku programu uruchomionego w interfejsie graficznym. Czasem program uruchomiony z poziomu menu graficznego nie zadziała. Jeśli wykorzystamy do tego celu wiersz poleceń, możemy zobaczyć komunikat o błędzie, informujący nas o problemie. Ponadto niektóre programy graficzne posiadają wiele ciekawych i przydatnych opcji wiersza poleceń.

Sygnały

Polecenie `kill` jest wykorzystywane do „zabicia" (zakończenia) procesu. Pozwala nam to na zakończenie wykonywania programu, który działa niepoprawnie lub nie chce zakończyć działania samoczynnie. Poniżej przedstawiony jest przykład:

```
[me@linuxbox ~]$ xlogo &
[1] 28401
[me@linuxbox ~]$ kill 28401
[1]+  Terminated              xlogo
```

Najpierw uruchamiamy w tle program `xlogo`. Powłoka wypisze numer zadania oraz PID procesu działającego w tle. Następnie używamy polecenia `kill` i podajemy PID procesu, który chcemy zakończyć. Zamiast numeru PID, moglibyśmy także użyć numeru zadania (na przykład %1).

Wygląda to na bardzo proste, ale to nie wszystko. Polecenie `kill` właściwie nie „zabija" procesów; raczej wysyła do nich sygnały. Sygnały stanowią jeden ze sposobów komunikacji systemu operacyjnego z programami. Działanie sygnałów poznaliśmy już przy korzystaniu z kombinacji klawiszy *Ctrl+C* i *Ctrl+Z*. Gdy użyjemy w terminalu jednej z tych kombinacji, terminal wyśle sygnał do programu pierwszoplanowego. W przypadku kombinacji *Ctrl+C* wysyłany jest sygnał `INT` (*Interrupt*); natomiast w przypadku kombinacji *Ctrl+Z* wysyłany jest sygnał `TSTP` (*Terminal Stop*). Z kolei programy „nasłuchują" sygnałów i mogą w odpowiedzi na nie wykonywać pewne działanie. Możliwość nasłuchiwania i działania w odpowiedzi na sygnały pozwala im na wykonywanie pewnych czynności, takich jak zapisywanie postępu pracy, gdy wysłany zostanie sygnał o zakończeniu.

Wysyłanie sygnałów do procesów za pomocą polecenia kill

Najczęściej stosowana składnia polecenia `kill` wygląda następująco:

```
kill [-signal] PID...
```

Jeśli nie sprecyzujemy żadnego sygnału w wierszu poleceń, to domyślnie wysłany zostanie sygnał `TERM` (*Terminate*). Polecenie `kill` jest najczęściej wykorzystywane do wysyłania sygnałów opisanych w tabeli 10.4.

Tabela 10.4. Często używane sygnały

Numer	Nazwa	Znaczenie
1	HUP	*Hung up*. Jest to pozostałość ze starych dobrych czasów, gdy terminale były podłączone do komputerów zdalnych poprzez kable telefoniczne i modemy. Sygnał jest wykorzystywany do wskazania programów, które terminal kontrolny „zawiesił". Efekt działania tego sygnału można zaobserwować poprzez zamknięcie sesji terminala. Program pierwszoplanowy działający w terminalu otrzyma sygnał i zakończy działanie. Sygnał ten jest też wykorzystywany przez wiele demonów do ponownej inicjalizacji. Gdy sygnał ten zostanie wysłany do demona, demon będzie uruchomiony ponownie i ponownie odczyta swój plik konfiguracyjny. Przykładem demona, który w ten sposób korzysta z sygnału HUP, jest serwer Apache.
2	INT	*Interrupt*. Wywołuje takie samo działanie jak kombinacja klawiszy *Ctrl+C* użyta w terminalu. Zwykle zatrzymuje działanie programu.
9	KILL	*Kill*. Jest to sygnał specjalny. Zwykle programy mogą obsługiwać wysyłane do nich sygnały na różne sposoby, a także je ignorować. Natomiast sygnał KILL nie jest tak naprawdę wysyłany do programu docelowego. W tym przypadku to jądro natychmiast kończy proces. Gdy proces zostanie w ten sposób zatrzymany, nie ma żadnej możliwości „posprzątania" po sobie lub zapisania efektów działania. Z tego powodu sygnału KILL należy używać tylko jako ostatniej deski ratunku, gdy inne sygnały służące do kończenia procesu zawiodą.
15	TERM	*Terminate*. Jest to domyślny sygnał wysyłany przez polecenie kill. Jeśli program jest nadal wystarczająco „żywy", aby przyjmować sygnały, zakończy swoje działanie.
18	CONT	*Continue*. Sygnał ten wznawia proces zatrzymany przez sygnał STOP.
19	STOP	*Stop*. Ten sygnał sprawia, że proces wstrzymuje działanie, lecz nie kończy pracy. Podobnie jak KILL, również ten sygnał nie jest przesyłany do procesu docelowego i dlatego nie może być zignorowany.

Wypróbujmy teraz działanie polecenia kill:

```
[me@linuxbox ~]$ xlogo &
[1] 13546
[me@linuxbox ~]$ kill -1 13546
[1]+  Hangup                  xlogo
```

W powyższym przykładzie uruchamiamy w tle program xlogo, a następnie wysyłamy do niego sygnał HUP poprzez polecenie kill. Program xlogo kończy działanie, a powłoka informuje, że proces w tle otrzymał sygnał *Hangup*. Możliwe, że zanim zobaczymy wiadomość, trzeba będzie kilkakrotnie nacisnąć klawisz *Enter*. Zauważmy, że sygnały można określić zarówno przez liczbę, jak i przez nazwę, włącznie z nazwą o przedrostku *SIG*:

```
[me@linuxbox ~]$ xlogo &
[1] 13601
[me@linuxbox ~]$ kill -INT 13601
```

```
[1]+  Interrupt               xlogo
[me@linuxbox ~]$ xlogo &
[1] 13608
[me@linuxbox ~]$ kill -SIGINT 13608
[1]+  Interrupt               xlogo
```

Powtórzmy powyższy przykład i wypróbujmy inne sygnały. Pamiętajmy, że zamiast numerów PID możemy skorzystać z numerów zadań.

Procesy, podobnie jak pliki, posiadają właścicieli. Jeśli jesteśmy właścicielem procesu (lub jeśli jesteśmy użytkownikiem uprzywilejowanym), możemy wysyłać do niego sygnały z wykorzystaniem polecenia `kill`.

Oprócz sygnałów wymienionych w tabeli 10.4, które są najczęściej wykorzystywane wraz z poleceniem `kill`, system często korzysta z innych sygnałów. W tabeli 10.5 opisano inne często używane sygnały.

Tabela 10.5. Inne często wykorzystywane sygnały

Numer	Nazwa	Znaczenie
3	QUIT	Wyjście.
11	SEGV	Naruszenie ochrony pamięci. Sygnał ten jest wysyłany, jeśli program korzysta z pamięci w nieuprawniony sposób, czyli gdy próbuje zapisać dane w miejscu, do którego nie jest uprawniony.
20	TSTP	Zatrzymanie terminala. Sygnał ten jest wysyłany przez terminal, gdy została użyta kombinacja *Ctrl+Z*. Inaczej niż w przypadku sygnału STOP, sygnał TSTP jest przechwytywany przez program, lecz może zostać zignorowany.
28	WINCH	Zmiana okna. Jest to sygnał wysyłany przez system podczas zmiany rozmiaru okna. Niektóre programy, takie jak `top` i `less`, będą odpowiadać na ten sygnał poprzez ponowne wyświetlenie wyników, aby dostosować się do nowych wymiarów okna.

Informacja dla ciekawskich — pełną listę sygnałów można wyświetlić, wykonując następujące polecenie:

```
[me@linuxbox ~]$ kill -l
```

Wysyłanie sygnałów do wielu procesów za pomocą polecenia killall

Możemy także wysyłać sygnały do wielu procesów, dotyczących określonego programu lub należących do określonego użytkownika. W tym celu stosujemy polecenie `killall`, którego składnia wygląda następująco:

```
killall [-u użytkownik] [-sygnał] nazwa...
```

Zademonstrujmy działanie tego polecenia. W tym celu uruchomimy kilka instancji programu `xlogo` i zakończymy ich działanie:

```
[me@linuxbox ~]$ xlogo &
[1] 18801
[me@linuxbox ~]$ xlogo &
[2] 18802
[me@linuxbox ~]$ killall xlogo
[1]-  Terminated              xlogo
[2]+  Terminated              xlogo
```

Pamiętajmy, że aby wysyłać sygnały do procesów, które do nas nie należą, musimy posiadać uprawnienia użytkownika uprzywilejowanego, podobnie jak w przypadku polecenia `kill`.

Więcej poleceń dotyczących procesów

Ponieważ monitorowanie procesów jest ważnym zadaniem administracyjnym, istnieje wiele poleceń do tego przeznaczonych. W tabeli 10.6 przedstawiono kilka poleceń wartych wypróbowania.

Tabela 10.6. Inne polecenia dotyczące procesów

Polecenie	Opis
`pstree`	Wyświetla procesy w postaci drzewa, pokazując zależności między procesami macierzystymi a potomnymi.
`vmstat`	Wyświetla informacje o wykorzystaniu zasobów systemu, włącznie z pamięcią, przestrzenią wymiany oraz dysku I/O. Aby wyświetlić stale uaktualnianą listę, należy podać po poleceniu czas opóźnienia (w sekundach) dla uaktualnienia (np. `vmstat 5`). Działanie programu kończymy, używając kombinacji *Ctrl+C*.
`xload`	Program graficzny rysujący wykres obciążenia systemu w czasie.
`tload`	Podobny do programu `xload`, jednak rysuje wykres w terminalu. Działanie programu kończymy, używając kombinacji *Ctrl+C*.

Skorowidz

D

E

F

G

H

I

J

K

L, Ł

M

N

O

P

R

S, Ś

T

U

V

W

X

Y

Z

TWOJA PRZEPUSTKA DO ŚWIATA ZAAWANSOWANYCH UŻYTKOWNIKÓW!

W latach dziewięćdziesiątych ubiegłego stulecia obsługiwanie komputera za pomocą komend wpisywanych w linii poleceń było całkiem naturalne. Ten stan rzeczy uległ zmianie wraz z wprowadzeniem systemu operacyjnego Windows 95, jednak wciąż wiele zadań łatwiej i szybciej można wykonać przy użyciu linii poleceń. Największy potencjał drzemie w linii poleceń systemu operacyjnego Linux. Przekonaj się, jak to wykorzystać i stać się wirtuozem klawiatury!

Jeżeli jesteś użytkownikiem systemu Linux i pracujesz z wykorzystaniem graficznego interfejsu użytkownika, to najwyższa pora poznać linię poleceń. Sięgnij po tę książkę i daj się ponieść eleganckim komendom, które zebrane w jedną całość potrafią zdziałać cuda. Na samym początku nauczysz się tworzyć pliki, katalogi oraz nawigować po zasobach dysku. W kolejnych rozdziałach przekonasz się, jak pomocne mogą być skrypty powłoki oraz w jaki sposób administrować systemem z poziomu linii poleceń. Ponadto opanujesz obsługę narzędzi do pobierania danych z serwerów FTP, wykorzystasz możliwość przekierowywania strumieni oraz dowiesz się, do czego służy edytor vi.

Ta książka wprowadzi Cię na kolejny poziom wtajemniczenia w obsłudze systemu Linux!

Dzięki tej książce:

- poznasz możliwości linii poleceń systemu Linux
- nauczysz się pisać skrypty automatyzujące pracę
- skorzystasz z zasobów FTP i WWW w trybie tekstowym
- opanujesz obsługę edytora vi

William E. Shotts jr — znawca systemu Linux. Od 15 lat pracuje z różnymi dystrybucjami oraz programuje w różnych językach. W branży IT działa od ponad 30 lat. Jest twórcą popularnego serwisu LinuxCommand.org.

Helion

32392 numer katalogowy

księgarnia internetowa
http://helion.pl

zamówienia telefoniczne
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-283-0174-0
9 788328 301740

cena: 79,00 zł

no starch press