Wydanie IV

Serwer Ubuntu

Kompletny przewodnik po Ubuntu Server 22.04

Jay LaCroix

Helion

Packt

Tytuł oryginału: Mastering Ubuntu Server: Explore the versatile, powerful Linux Server distribution Ubuntu 22.04 with this comprehensive guide, 4th Edition

Tłumaczenie: Magdalena A. Tkacz

ISBN: 978-83-8322-592-0

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/serub4
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

Rozdział 7

Procesy — kontrolowanie i zarządzanie

Na typowym serwerze linuksowym w dowolnym momencie może być uruchomionych ponad sto procesów. Są to zarówno procesy związane z usługami systemowymi, takimi jak **NTP** (ang. *Network Time Protocol*), jak i procesy, które dostarczają na zewnątrz informacje — takie jak serwer internetowy Apache. Jako administrator serwerów Ubuntu musisz umieć zarządzać tymi procesami, a także zarządzać zasobami dla nich dostępnymi. W tym rozdziale przyjrzymy się zarządzaniu procesami, w tym poleceniu `ps`, zarządzaniu komendami kontroli zadań i innymi kwestiami.

W miarę omawiania kolejnych pojęć będę omawiać następujące tematy:

- Zarządzanie zadaniami.
- Korzystanie z polecenia `ps`.
- Zmienianie priorytetu procesów.
- Radzenie sobie z niewłaściwie działającymi procesami.
- Zarządzanie procesami systemowymi.
- Planowanie zadań za pomocą `cron`.

Aby rozpocząć naszą przygodę z zarządzaniem procesami, przyjrzyjmy się najpierw zarządzaniu zadaniami. Nie tylko pomoże nam to lepiej zrozumieć ich ideę, ale także pozwoli lepiej zrozumieć działanie procesów w tle i procesów pierwszoplanowych.

Zarządzanie zadaniami

Do tej pory wszystko, co robiliśmy na powłoce, od momentu uruchomienia polecenia do jego zakończenia, było widoczne na ekranie. Instalowaliśmy aplikacje, uruchamialiśmy programy i wykonywaliśmy różne polecenia. Za każdym razem podczas wykonywania zadania „traciliśmy" na chwilę kontrolę nad powłoką — mogliśmy zacząć robienie czegoś nowego dopiero po zakończeniu wykonywania poprzedniego. Na przykład, gdy

chcieliśmy zainstalować pakiet *vim-nox* za pomocą polecenia `apt install`, musieliśmy patrzeć bezczynnie, jak `apt` pobiera pakiet i instaluje go za nas.

Podczas wykonywania zadania, gdy powłoka wykonuje dla nas zlecone jej zadania, nie pozwala nam na wydanie innego polecenia. Co prawda zawsze możemy na serwerze otworzyć nową powłokę i korzystać z wielozadaniowości, mając otwarte jednocześnie dwa okna i w każdym wykonując inne zadania. Ale nie jest to najbardziej efektywna metoda korzystania z wielozadaniowości podczas pracy z wierszem poleceń.

Zamiast tego możemy zdecydować o wykonywaniu procesu w tle, bez czekania na jego zakończenie, a w międzyczasie pracować nad czymś innym. Następnie możemy wywołać z powrotem proces, aby albo dalej z nim pracować, albo sprawdzić, czy jego wykonanie zakończyło się pomyślnie. Pomyśl o tym jako o czymś podobnym do środowiska pulpitu z oknami lub interfejsów użytkownika w systemach operacyjnych Windows lub macOS. Możesz pracować z aplikacją, zminimalizować ją, aby przestała być widoczna, a następnie zmaksymalizować, aby dalej z nią pracować. Generalnie chodzi o to samo, o co chodzi w przypadku działania procesu w tle w powłoce Linuksa.

Więc jak spowodować, by proces działał w tle, a jak, by działał na pierwszym planie? Może to być problematyczne do wyjaśnienia. Moim zdaniem najłatwiejszym sposobem na nauczenie się tego jest wypróbowanie, jak to działa. Najłatwiejszym przykładem, jaki przychodzi mi do głowy, jest (po raz kolejny) użycie edytora tekstu. Obiecuję, że tym razem użycie edytora tekstu w roli przykładu nie będzie nudne. Ten przykład jest akurat niezwykle przydatny i możliwe, że stanie się rutynowym elementem, z którego będziesz korzystał na co dzień w pracy. Aby wykonać to ćwiczenie, możesz użyć dowolnego, preferowanego przez siebie edytora tekstu działającego w wierszu poleceń, takiego jak Vim lub nano. Na serwerze Ubuntu nano jest zwykle zainstalowany domyślnie, więc już go masz, jeśli chcesz z niego korzystać. Jeśli wolisz używać Vim, nie obawiaj się i zainstaluj pakiet *vim-nox* (jeśli jeszcze tego nie zrobiłeś):

```
sudo apt install vim-nox
```

Możesz zainstalować Vim zamiast *vim-nox*, ale ja zawsze decyduję się na *vim-nox*, ponieważ ma on wbudowaną obsługę języków skryptowych.

Jednak nie ma problemu, jeśli chcesz używać dowolnego edytora tekstu, z którym czujesz się komfortowo. W poniższych przykładach będę używał nano, ale jeśli używasz Vim, po prostu zamień nazwę `nano` na `vim` za każdym razem, gdy ją zobaczysz.

Tak czy inaczej, aby zobaczyć działanie w tle w akcji, otwórz swój edytor tekstu. Możesz otworzyć jakiś plik lub po prostu rozpocząć pustą sesję. (Jeśli masz wątpliwości, wpisz **`nano`** i naciśnij *Enter*). Mając otwarty edytor tekstu, możemy w każdej chwili „zlecić mu", by działał w tle, naciskając *Ctrl+Z* na klawiaturze.

Jeśli używasz Vim zamiast nano, możesz zlecić mu przejście do pracy w tle tylko wtedy, gdy jesteś w trybie poleceń, ponieważ tylko wtedy *Ctrl+Z* zostanie przekazane do powłoki.

Czy widziałeś, co się stało? Edytor znikł z ekranu i powróciłeś do powłoki, więc możesz teraz dalej wykonywać polecenia. Powinieneś zobaczyć na wyjściu komunikat podobny do poniższego:

```
[1]+ Stopped nano
```

Widzimy tutaj numer **zadania** (ang. *job*) naszego procesu, jego status, a następnie nazwę procesu. Nawet jeśli proces Twojego edytora tekstu pokazuje status `Stopped`, to nadal jest uruchomiony. Możesz to potwierdzić za pomocą następującego polecenia:

```
ps au | grep nano
```

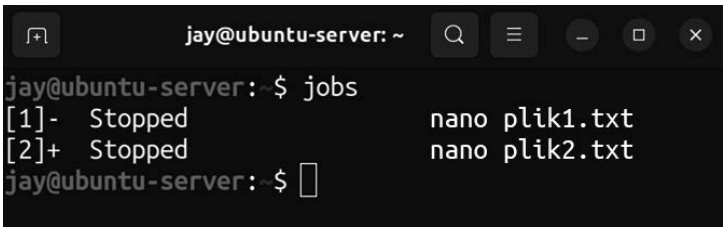
W moim przypadku widzę, że proces `nano` działa z identyfikatorem procesu (PID) o numerze `43231`:

```
jay 43231 0.0 0.1 5468 3632 pts/0 T 11:27 0:00 nano
```

W tym momencie mogę wykonywać dodatkowe polecenia, poruszać się po moim systemie plików i wykonywać inną pracę. Kiedy chcę przywrócić mój edytor tekstu, mogę użyć polecenia `fg`, aby przywrócić proces na pierwszy plan, co spowoduje wznowienie pracy edytora. Jeśli mam wiele procesów w tle, polecenie `fg` przywróci ten, z którym ostatnio pracowałem.

Użyłem polecenia `ps`, aby pokazać Ci, że proces nadal działa w tle, ale w rzeczywistości istnieje dedykowane polecenie do tego celu, a jest nim polecenie `jobs`.

Jeśli wydasz polecenie `jobs`, zobaczysz na wyjściu listę wszystkich procesów działających w tle (rysunek 7.1).



Rysunek 7.1. Uruchamianie polecenia jobs po przesłaniu do pracy w tle dwóch procesów nano

Na wyjściu widać, że korzystam z dwóch sesji `nano`, w jednej mam otwarty *plik1.txt*, a w drugiej *plik2.txt*. Gdybym wykonał polecenie `fg`, przywołałbym sesję `nano`, w której mam otwarty *plik2.txt*, ponieważ była to ostatnia sesja, w której pracowałem. To może, ale nie musi być ten plik, do którego edycji chcę wrócić. Ponieważ po lewej stronie

mam identyfikator zadania, to używając tego identyfikatora, mogę za pomocą polecenia `fg` przywołać określony proces działający w tle:

```
fg 1
```

Umiejętność przełączania procesów na działanie w tle może usprawnić wykonywanie przez Ciebie zadań w pracy. Na przykład załóżmy, że edytujesz plik konfiguracyjny dla aplikacji serwerowej, takiej jak Apache. Podczas edycji tego pliku konfiguracyjnego musisz sprawdzić dokumentację (stronę man) dla Apache, ponieważ zapomniałeś składni jakiegoś polecenia. Mógłbyś otworzyć nową powłokę i nową sesję SSH do serwera i przeglądać dokumentację w innym oknie. Ale jeśli otworzysz zbyt wiele powłok, może zrobić się zamieszanie. O wiele prościej jest przełączyć do działania w tle bieżącą sesję `nano`, przeczytać dokumentację, a następnie przywrócić na pierwszy plan proces poleceniem `fg` i kontynuować w nim pracę, i to wszystko w ramach jednej sesji SSH!

Aby proces działał w tle, nie musisz używać *Ctrl+Z*; właściwie to możesz ustawić proces, aby działał w tle od razu po uruchomieniu; wystarczy wpisać polecenie z symbolem *ampersand* (*&*) na końcu. Aby pokazać, jak to działa, użyję jako przykładu `htop`. Trzeba przyznać, że nie jest to najbardziej praktyczny przykład, ale działa, i wystarcza do pokazania, jak uruchomić proces od razu w tle.

Być może nie masz jeszcze zainstalowanego pakietu `htop` — w tym wypadku zachęcam do jego zainstalowania, a następnie uruchomienia go z symbolem *&*:

```
sudo apt install htop
htop &
```

Pierwsza komenda, jak już wiesz, instaluje pakiet `htop` na naszym serwerze. Drugim poleceniem uruchamiam `htop`, ale natychmiast przełączam go do działania w tle. Tym, co zobaczę, gdy będzie działał w tle, jest jego identyfikator zadania i identyfikator procesu (więcej na ten temat w następnym rozdziale). Teraz w każdej chwili mogę przywrócić `htop` na pierwszy plan za pomocą `fg`. Ponieważ właśnie przełączyłem go, by działał w tle, `fg` go przywróci, ponieważ uważa go za ostatni proces, który został przełączony do działania w tle. Jak już wiesz, aby go przywrócić, gdyby nie był ostatnim, mógłbym odwołać się do ID jego zadania za pomocą polecenia `fg`. Dobrze by było, gdybyś teraz poćwiczył używanie poleceń z symbolem *&*, a następnie przywracanie procesu na pierwszy plan. W przypadku `htop` użyteczne może być uruchomienie go, przełączenie do działania w tle, a następnie przywrócenie na pierwszy plan w momencie, gdy potrzebujesz sprawdzić wydajność swojego serwera.

Pamiętaj jednak, że po opuszczeniu powłoki wszystkie procesy działające w tle zostaną zamknięte. Jeśli masz niezapisaną pracę w edytorze tekstu, stracisz to, nad czym pracowałeś. Z tego powodu, jeśli używasz procesów w tle, możesz sprawdzić, czy masz jakieś oczekujące zadania, wykonując polecenie `jobs` (dosł. zadania), zanim się wylogujesz.

Prawdopodobnie zauważysz, że niektóre aplikacje bezproblemowo działają w tle, podczas gdy inne nie. W przypadku używania edytora tekstu i `htop` te aplikacje po przełączeniu na działanie w tle zostają wstrzymane (ang. *paused*), pozwalając nam na wykonywanie innych zadań i późniejszy powrót do tych poleceń. Jednak niektóre aplikacje niezależnie od tego, czy działają w tle, czy nie, mogą nadal regularnie wyświetlać diagnostyczny tekst w Twoim głównym oknie. Aby uzyskać jeszcze większą kontrolę nad sesjami Bash, możesz nauczyć się używać multipleksera, takiego jak *tmux* lub *screen*, aby umożliwić tym procesom uruchamianie się w ich własnych sesjach, tak by nie przeszkadzały Ci w pracy. Używanie programu takiego jak *tmux* wykracza poza zakres tej książki, ale jest to przydatne narzędzie, które warto poznać.

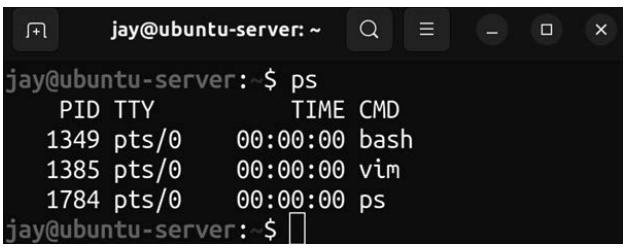
Możliwość ustawienia pracy procesu w tle i na pierwszym planie pozwala nam na bardziej efektywne zarządzanie zadaniami z linii poleceń i jest zdecydowanie przydatne. Teraz możemy szerzej spojrzeć na przeglądanie procesów na serwerze, w tym tych, które nie są przez nas uruchamiane ręcznie, jak to miało miejsce w przypadku edytora tekstu. W następnej sekcji przyjrzymy się poleceniu `ps`, które może pomóc w zrozumieniu, co tak naprawdę jest uruchomione na naszym serwerze.

Polecenie ps

Podczas zarządzania serwerem musisz rozumieć, jakie procesy są uruchomione i jak nimi zarządzać. W dalszej części tego rozdziału zajmiemy się uruchamianiem, zatrzymywaniem i monitorowaniem procesów. Jednak zanim przejdziemy do tych koncepcji, musisz najpierw być w stanie określić, co tak naprawdę jest uruchomione na serwerze. Do tego służy polecenie `ps`.

Wyświetlanie uruchomionych procesów za pomocą ps

Polecenie `ps` wywołane bez parametrów pokaże listę procesów uruchomionych przez użytkownika, który wywołał polecenie (rysunek 7.2).



Rysunek 7.2. Wyjście polecenia ps po uruchomieniu jako zwykły użytkownik i bez żadnych opcji

Na rysunku 7.2 widać, że gdy uruchomiłem polecenie `ps` na koncie mojego użytkownika bez żadnych opcji, pokazało mi ono listę procesów, które uruchamiam jako ja sam. W tym przypadku mam otwartą sesję `vim` (działającą w tle), a w ostatnim wierszu widzimy również samo `ps`, które jest również pokazywane.

Po lewej stronie wyświetlonych informacji zobaczysz numer każdego z uruchomionych procesów. Jest to tzw. **identyfikator procesu** (ang. *process ID*, PID), o którym wspominaliśmy w podrozdziale „Zarządzanie zadaniami". PID jest czymś, co naprawdę powinieneś znać, więc równie dobrze możemy go omówić teraz, zanim przejdziemy dalej.

Każdy proces uruchomiony na serwerze ma przypisany PID, który odróżnia go od innych procesów w systemie. Możesz rozróżniać procesy po nazwach — jako `vim` lub `top` (lub pod jakimikolwiek innymi nazwami). Jednak serwer rozróżnia procesy po ich ID. Kiedy otwierasz program lub uruchamiasz proces, otrzymuje on PID od jądra. Podczas pracy nad zarządzaniem serwerem przekonasz się, że znajomość PID jest przydatna, zwłaszcza w przypadku poleceń, które omówimy w tym rozdziale. Jeśli chcesz na przykład zabić źle działający proces, typowym sposobem postępowania będzie znalezienie PID tego procesu, a następnie odwołanie się do niego, gdy wydajesz polecenie, by go zabić (jak to zrobić, pokażę w dalszej części rozdziału). W rzeczywistości PID jest bardziej złożony niż tylko reprezentujący go numer przypisany do uruchomionego procesu, ale na potrzeby tego rozdziału wystarczy, że zapamiętasz, iż proces jest identyfikowany jako numer.

Jeśli znasz nazwę procesu, możesz również użyć polecenia `pidof`, aby znaleźć PID procesu. Na przykład pokazałem Ci zrzut ekranu procesu `vim` uruchomionego z PID o wartości 1385. Możesz również odczytać jego PID, uruchamiając następujące polecenie:

```
pidof vim
```

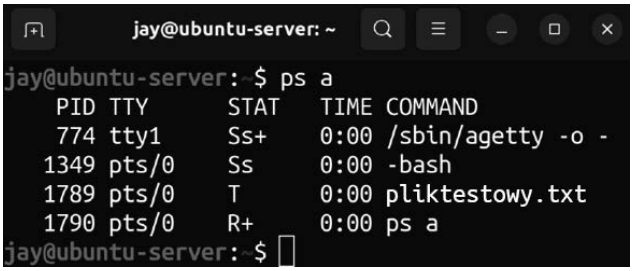
W odpowiedzi otrzymasz PID procesu bez konieczności używania polecenia `ps`.

Opcje dla ps

Pracując z poleceniem `ps`, warto znać kilka użytecznych opcji, które możesz dodać, aby zmienić sposób, w jaki `ps` wyświetla dane na wyjście. Jeśli użyjesz opcji `a`, zobaczysz więcej informacji niż dla samego `ps`:

```
ps a
```

Spowoduje to wyświetlenie danych wyjściowych pokazanych na rysunku 7.3.



Rysunek 7.3. Wynik po wydaniu polecenia ps a

Po użyciu polecenia `ps a` widzimy to samo co poprzednio, ale dodatkowo mamy informacje oraz nagłówki kolumn na górze. Teraz widzimy nagłówki dla `PID`, `TTY`, `STAT`, `TIME` i `COMMAND`. Dzięki temu możesz zobaczyć, że procesy `vim`, które mam uruchomione, edytują plik o nazwie *pliktestowy.txt*. Jest to świetna informacja, ponieważ jeśli miałbym otwartych więcej sesji `vim` i jedna z nich zachowywałaby się niewłaściwie, prawdopodobnie chciałbym wiedzieć, którą z nich konkretnie muszę zatrzymać.

Widzieliśmy już pola `PID` i `COMMAND`, choć nie widzieliśmy formalnego nagłówka u góry. `PID` został już omówiony, więc nie będę wchodził w dodatkowe szczegóły na ten temat. Pole `COMMAND` informuje nas o aktualnie wykonywanym poleceniu, co jest bardzo przydatne, jeśli chcemy się upewnić, że zarządzamy właściwym procesem, lub zobaczyć, co uruchamia konkretny użytkownik (wkrótce pokażę, jak wyświetlić procesy innych użytkowników).

Pole `STAT` jest czymś nowym; nie widzieliśmy go podczas uruchamiania `ps` bez opcji. Pole `STAT` to kod statusu procesu, który odnosi się do tego, w jakim stanie aktualnie znajduje się proces. Stanem tym może być permanentne uśpienie (`D`), nieaktywny (`Z`), zatrzymany (`T`), drzemka (`S`) oraz w kolejce do uruchomienia (`R`). Jest jeszcze stronicowanie (`W`), ale ponieważ nie jest już używane, nie widzę potrzeby, by je omawiać. Permanentne uśpienie to stan, w którym zasadniczo proces oczekuje na dane na wejście i nie może obsługiwać dodatkowych sygnałów (o sygnałach krótko powiem w dalszej części tego rozdziału). Proces nieaktywny (określany również jako *proces zombie*), to proces, który wykonał wszystko, co miał do wykonania, zakończył swoją pracę, ale czeka na proces macierzysty, aby zostać usuniętym. Nieaktywne procesy de facto nie działają — nie są uruchomione, ale jeszcze pozostają na liście procesów i normalnie powinny się same zakończyć. Jeśli taki proces widnieje stale na liście i nie zamyka się, może być kandydatem do użycia do niego polecenia `kill` (zabij), które omówimy później. Taki proces to na ogół proces, który został przełączony do pracy w tle — omówię to w następnym rozdziale. Przerywalne uśpienie oznacza, że program jest bezczynny: czeka na dane na wejściu, które go wybudzą.

Kolumna `TTY` mówi nam, do którego TTY dany proces jest podłączony. TTY odnosi się do **dalekopisu** (ang. *teletypewriter*), który jest terminem pochodzącym z innych czasów.

W przeszłości, w czasach wielkich komputerów mainframe, użytkownicy korzystali z takich komputerów za pomocą „terminali" — urządzeń składających się z monitora i klawiatury, podłączonych (przez kabel) do mainframe. Takie urządzenia mogły jedynie wyświetlać dane wyjściowe otrzymywane z komputera mainframe i odbierać dane wpisywane na klawiaturze. Teletypewriter był terminem stosowanym w odniesieniu do takich urządzeń. Oczywiście obecnie nie używamy takich maszyn, ale koncepcja działania jest podobna.

Na serwerze używamy klawiatury do wysyłania danych wejściowych do urządzenia, które następnie wyświetla nasze dane wyjściowe innemu urządzeniu. W tym przypadku urządzeniem wejściowym jest nasza klawiatura, a urządzeniem wyjściowym jest nasz ekran, który jest albo bezpośrednio podłączony do serwera, albo znajduje się na naszym komputerze połączonym z naszym serwerem poprzez usługę taką jak SSH. W systemie Linux większość procesów działa na TTY, który jest (praktycznie rzecz biorąc) terminalem. Przechwytuje on dane wejściowe i zarządza wyjściowymi, podobnie jak to miało miejsce w dalekopisie. Terminal jest metodą, którą wykorzystujemy do interakcji z serwerem.

Na rysunku 7.3 mamy proces działający na TTY o nazwie `tty1`, a pozostałe procesy działają na `pts/0`. TTY które widzimy, to rzeczywiste urządzenia terminalowe, a `pts` odnosi się do wirtualnego (pseudo-) urządzenia terminalowego. Nasz serwer jest w stanie uruchomić kilka sesji `tty`, zazwyczaj od jednej do siedmiu. Każda z nich może uruchamiać swoje własne programy i procesy. Aby to lepiej zrozumieć, spróbuj wcisnąć *Ctrl+Alt+* dowolny klawisz funkcyjny, od *F1* do *F7* (jeśli masz fizyczną klawiaturę podłączoną do fizycznego serwera). Za każdym razem powinieneś zobaczyć czysty ekran, każdy przeniesiony na inny terminal. Każdy z tych terminali działa niezależnie od innych. Każdy z Twoich klawiszy funkcyjnych reprezentuje konkretny TTY, więc naciskając *Ctrl+Alt+F6*, widzisz na ekranie to, co jest na TTY 6.

Zasadniczo możesz przełączać się między TTY od TTY 1 aż do TTY 7, każdy może mieć swoje własne uruchomione procesy. Jeśli ponownie uruchomisz `ps a`, zobaczysz, że wszystkie procesy uruchomione na tych TTY pokażą się na wyjściu jako sesje `tty`, takie jak `tty2` lub `tty4`. Procesy uruchamiane w emulatorze terminala otrzymają oznaczenie `pts`, ponieważ nie są uruchamiane w rzeczywistym TTY, ale raczej w pseudo-TTY.

To był dość długi wywód o czymś, co prowadzi do prostego rozróżnienia (TTY lub pseudo-TTY), ale z tą wiedzą powinieneś być w stanie odróżnić, który proces jest uruchomiony na serwerze, a który przez powłokę.

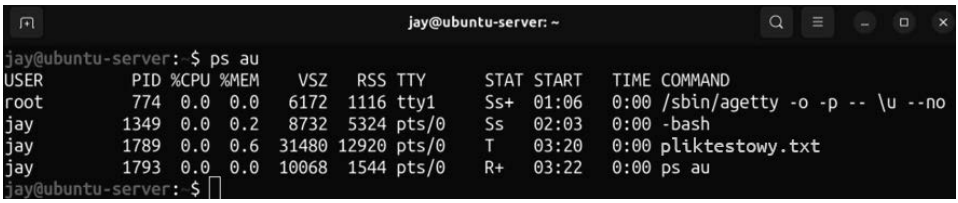
W ramach kontynuacji opisu kolumn z rysunku 7.3 spójrzmy na pole `TIME`. Pole to reprezentuje całkowity czas, przez jaki dany proces wykorzystywał procesor. Jednakże na zrzucie ekranu, który pokazałem, czas wynosi `0:00` dla każdego z procesów. Na początku może to być mylące. W moim przypadku procesy obsługujące `vim` były uruchomione

około 15 minut przed zrobieniem zrzutu ekranu, ale mimo to nadal pokazują czas wykorzystania `0:00`. Powodem jest to, że to nie jest czas działania procesu, ale raczej czas, w którym proces aktywnie angażuje procesor. W przypadku `vim` każdy z procesów to po prostu bufor z otwartym plikiem. Dla porównania na maszynie z systemem Linux, na której piszę ten rozdział, jest proces o identyfikatorze ID `759` z czasem `92:51`. PID `759` jest procesem mojego *serwera X* (ang. *X server*), który jest podstawą mojego **graficznego interfejsu użytkownika** (ang. *graphical user interface*, **GUI**) odpowiadającego za możliwość pracy w środowisku okienek. Jednak ten laptop pracuje obecnie od 6 dni i 22 godzin, co w przybliżeniu odpowiada 166 godzinom, a więc nie jest to ten sam czas, który PID `759` zgłasza w swoim wpisie `TIME`. Możemy zatem z tego wywnioskować, że mimo iż mój laptop działał przez 6 dni z rzędu, serwer X wykorzystał jedynie 92 godziny i 51 minut rzeczywistego czasu pracy procesora. Podsumowując, kolumna `TIME` odnosi się do czasu, w którym proces potrzebuje procesora, aby coś obliczyć, i niekoniecznie pokazywana w niej wartość jest równa temu, jak długo coś jest uruchomione lub jak długo proces graficzny jest wyświetlany na ekranie.

Wróćmy do polecenia `ps` i spójrzmy na kilka dodatkowych opcji. Najpierw zobaczmy, co otrzymamy, gdy do naszego poprzedniego przykładu dodamy opcję `u`, co daje nam następujące przykładowe polecenie:

```
ps au
```

Spowoduje to wygenerowanie informacji podobnych do pokazanych na rysunku 7.4.



Rysunek 7.4. Wynik działania polecenia ps au

Kiedy uruchomisz to polecenie, powinieneś od razu zauważyć różnicę w stosunku do polecenia `ps a`. Dzięki dodaniu tej opcji zobaczysz na liście procesy wraz z dodatkową informacją będącą identyfikatorami użytkowników, którzy je uruchomili. Uruchamiając polecenie z tą opcją, widzę procesy mojego użytkownika (`jay`), jak również jeden dla użytkownika *root*. Opcji `u` będziesz często używać, ponieważ przez większość czasu, kiedy zarządzasz serwerami, prawdopodobnie bardziej niż przyglądanie się procesom systemowym interesuje Cię obserwowanie, co kombinują użytkownicy Twojego serwera. Ale prawdopodobnie najczęstszym zastosowaniem polecenia `ps` jest jego następująca odmiana:

```
ps aux
```

Po dodaniu opcji `x` nie ograniczamy już naszego wyjścia do procesów w obrębie TTY (czy to natywnych, czy pseudo). W rezultacie zobaczymy dużo więcej procesów, w tym procesy systemowe, które nie są związane z procesem, który sami uruchomiliśmy. Bardzo proszę, wypróbuj to. W praktyce polecenie `ps aux` jest jednak najczęściej używane z `grep` do wyszukania konkretnego procesu lub ciągu znaków. Załóżmy, że chcesz zobaczyć listę wszystkich procesów roboczych `nginx`. Aby to zrobić, możesz wydać następujące polecenie:

```
ps aux | grep nginx
```

Tutaj wykonujemy tak jak poprzednio polecenie `ps aux`, ale przesyłamy wyjście do `grep`, gdzie szukamy tylko tych linii wyjścia, które zawierają ciąg `nginx`. W praktyce jest to sposób, w jaki zarówno ja, jak i inni administratorzy (na ile zauważyłem) często używamy `ps`. Używając `ps aux`, jesteś w stanie zobaczyć dużo więcej danych wyjściowych, a następnie możesz zawęzić wyniki za pomocą kryteriów wyszukiwania poprzez przesłanie strumienia (ang. *piping*) do `grep`. Jeśli jednak chcesz zobaczyć tylko te procesy, które zawierają określony ciąg znaków, możesz to również osiągnąć za pomocą polecenia:

```
ps u -C nginx
```

W ten sposób otrzymamy wynik zawierający listę procesów pasujących do `nginx` oraz związane z nimi szczegóły. Inną użyteczną odmianą polecenia `ps` jest posortowanie wyniku poprzez malejące posortowanie procesów z wykorzystaniem CPU jako kryterium:

```
ps aux --sort=-pcpu
```

Niestety, polecenie to pokazuje wiele danych wyjściowych i musielibyśmy przewijać je do góry, aby zobaczyć najważniejsze procesy. W zależności od ustawień Twojego terminala możesz nie mieć możliwości przewijania wyników do początkowych (lub przewijania w ogóle), więc poniższa komenda zawęzi je jeszcze bardziej:

```
ps aux --sort=-pcpu | head -n 5
```

Teraz wynik jest przydatny! W tym przykładzie używam polecenia `ps aux` z opcją `-sort`, sortując według procentowego wykorzystania procesora (`-pcpu`). Następnie przekazuję wyjście do polecenia `head`, gdzie instruuję je, aby pokazało mi tylko pięć linii (`-n 5`). W efekcie otrzymuję listę pięciu procesów, które zużyły najwięcej mocy procesora od czasu startu systemu. Mogę zrobić to samo, sprawdzając wykorzystanie pamięci:

```
ps aux --sort=-pmem | head -n 5
```

Jeśli chcesz sprawdzić, które procesy zachowują się niewłaściwie i używają nietypowej ilości pamięci lub procesora, te polecenia pomogą Ci zawęzić listę. Polecenie `ps` jest bardzo użytecznym poleceniem w zestawie narzędzi administratora. Nie krępuj się eksperymentować z nim, nie ograniczając się do przykładów, które podałem; możesz

zajrzeć na strony *man* polecenia `ps`, aby nauczyć się jeszcze więcej sztuczek. W drugiej sekcji strony *man* dla `ps` — pod *Examples* (przykłady) — znajdziesz nawet więcej ciekawych przykładów do wypróbowania.

Teraz gdy wiesz już, jak sprawdzać uruchomione procesy, w następnej części przyjrzymy się temu, jak zmienić priorytety procesów, aby procesor mógł się wydajniej zająć tymi, które są ważniejsze.

Zmienianie priorytetu procesów

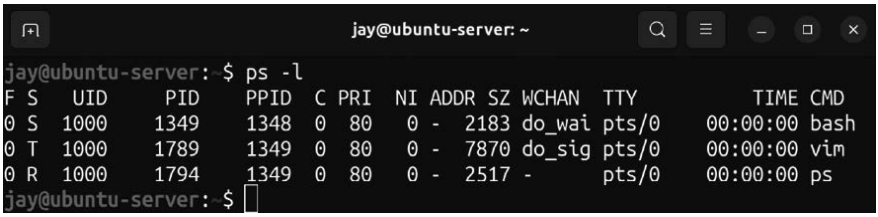
Procesy w systemie Linux mogą być uruchamiane z priorytetem (który można zmienić), co daje niektórym procesom większy priorytet, a innym mniejszy. Daje też to Tobie, jako administratorowi, pełną swobodę, jeśli chodzi o zapewnienie, że najważniejsze procesy w systemie będą uruchamiane z priorytetem na odpowiednim poziomie. Do tego celu służą specjalne polecenia: `nice` i `renice`. Polecenia te pozwalają odpowiednio: uruchomić proces o określonym priorytecie lub zmienić priorytet już uruchomionego procesu.

W dzisiejszych czasach ręczna edycja priorytetu procesu jest czymś, co administratorzy robią rzadziej niż kiedyś. Procesor z 32 rdzeniami (lub jeszcze więcej) nie jest rzadkością, podobnie jak setki gigabajtów pamięci RAM. Dzisiejsze serwery są z pewnością bardziej wydajne niż kiedyś, a przy tym nie „cierpią” na niedostatek zasobów sprzętowych jak dawniej. Wiele serwerów (uruchomionych jako maszyny wirtualne) i kontenerów jest dedykowanych do pojedynczego zadania, więc dostrajanie priorytetów procesów straciło na znaczeniu. Jednak firmy przetwarzające dane i wykorzystujące funkcje uczenia głębokiego mogą znaleźć się w sytuacji, w której będą musiały dostroić niektóre parametry.

Niezależnie od tego, czy nadawanie priorytetów procesom jest czymś, z czego natychmiast skorzystasz, czy nie, dobrze by było rozumieć tę ideę — na wypadek gdybyś kiedyś potrzebował zwiększyć lub zmniejszyć priorytet jakiegoś procesu. Powróćmy do polecenia `ps`, tym razem z argumentem `-l`:

```
ps -l
```

Dane wyjściowe po wydaniu tego polecenia będą wyglądały tak jak na rysunku 7.5.



Rysunek 7.5. Wynik po wydaniu polecenia ps -l

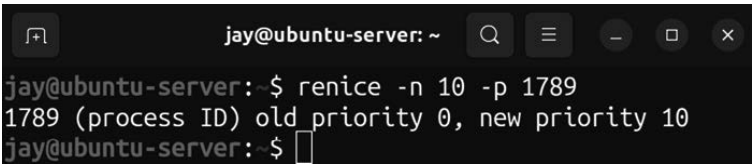
W wynikach, które pojawiły się po użyciu polecenia `ps -l`, możesz zobaczyć kolumny `PRI` i `NI`. `PRI` odnosi się do priorytetu (ang. *priority*), a `NI` odnosi się do wartości parametru określającego „uprzejmość" (ang. *niceness*), którą omówię bardziej szczegółowo w dalszej części tego rozdziału. W tym przykładzie każdy proces ma `PRI` równe `80` i `NI` równe `0`. Nie modyfikowałem żadnej z tych wartości; są to wartości, które otrzymuję, gdy uruchamiam procesy bez żadnych specjalnych zmian. Wartość `PRI` równa `80` jest domyślną wartością początkową dla wszystkich procesów i będzie się zmieniać, gdy będziemy zwiększać lub zmniejszać wartość parametru uprzejmości.

Jak już wspomniałem, mamy dedykowane komendy, które pozwalają nam zmieniać priorytety, mianowicie `nice` i `renice`. To, której z nich użyć, zależy od tego, czy proces jest już uruchomiony, czy nie. Jeśli chodzi o procesy wymienione na rysunku 7.5, trzeba użyć `renice`, aby zmienić ich priorytet, ponieważ wszystkie są już uruchomione. Jeśli chcielibyśmy uruchomić proces i nadać mu od początku określony priorytet, użylibyśmy `nice`.

Na przykład zmodyfikujmy proces sesji `vim`, którą mam uruchomioną. Oczywiście, jest to trochę kiepski przykład, ponieważ `vim` nie jest bardzo ważnym procesem. W prawdziwym świecie nadawałbyś priorytety procesom, które są rzeczywiście ważne. W moim przypadku, ponieważ proces `vim` ma PID o wartości `1789`, polecenie, które musiałbym uruchomić, aby zmienić „uprzejmość" procesu, wyglądałoby tak:

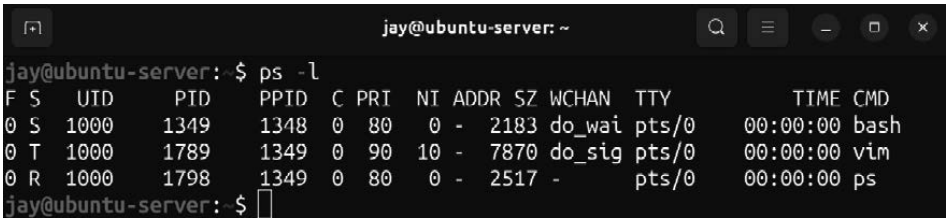
```
renice -n 10 -p 1789
```

Dane wyjściowe po wykonaniu tego polecenia będą wyglądały jak na rysunku 7.6.



Rysunek 7.6. Zmiana priorytetu procesu za pomocą renice

Jeśli uruchomisz ponownie `ps -l`, zobaczysz nową, ładną wartość dla `vim` (rysunek 7.7).

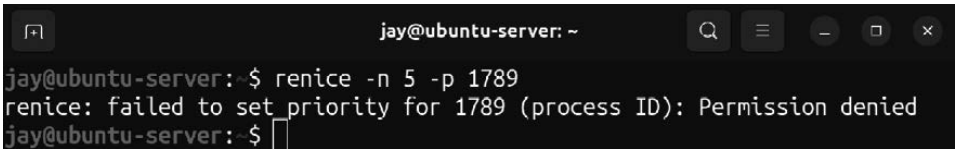


Rysunek 7.7. Wyjście polecenia ps -l po zmianie priorytetu procesu

Teraz dla `vim` mamy nową wartość uprzejmości, jest ona równa `10` i pojawia się pod `NI`, a wartość `PRI` wzrosła do `90`. Od teraz ta instancja `vim` będzie działać z niższym priorytetem niż inne moje zadania: im wyższa wartość uprzejmości, tym niższy priorytet. Zauważ, że kiedy zmieniłem priorytet tym poleceniem, nie użyłem `sudo`. W tym przykładzie jest to w porządku, ponieważ zwiększam wartość uprzejmości procesu, a to jest dozwolone. Spróbuję jednak zmniejszyć wartość uprzejmości bez `sudo`, używając następującego polecenia:

```
renice -n 5 -p 1789
```

Jak widać na rysunku 7.8, to już się nie udaje.



Rysunek 7.8. Próba zmniejszenia priorytetu dla procesu

Moja próba zmniejszenia wartości uprzejmości z 10 do 5 została zablokowana. Gdybym był w stanie obniżyć uprzejmość, wtedy spowodowałbym, że mój proces zacząłby działać z wyższym priorytetem. Zamiast tego otrzymałem jednak komunikat o błędzie *Permission denied*. Tak więc widać, że zasadniczo użytkownicy mogą zwiększać uprzejmość swoich procesów, ale nie mogą jej zmniejszać, nawet dla procesów, które sami zainicjowali. Jeśli chcesz zmniejszyć wartość uprzejmości, musisz to zrobić, używając `sudo`. Więc zasadniczo, jeśli chcesz być „bardziej uprzejmy", możesz to zrobić. Jeśli chcesz być „bardziej wredny", będziesz potrzebował uprawnień użytkownika *root*. Na dodatek użytkownik nie może zmienić priorytetu procesu, którego nie jest właścicielem. Jeśli więc spróbujesz użyć `renice` do zmiany priorytetu zadania uruchomionego przez innego użytkownika, otrzymasz komunikat o błędzie: *Operation not permitted*.

W tym momencie wiesz już, jak zmienić priorytet uruchomionych procesów za pomocą `renice`. Teraz zobacz, jak za pomocą `nice` uruchomić nowy proces z określonym priorytetem. Spójrz na następujące polecenie:

```
nice -n 10 vim
```

Tutaj uruchamiamy nową instancję `vim`, ale od razu z priorytetem ustawionym na konkretną wartość. Jeśli chcemy później ponownie zmienić priorytet `vim`, będziemy musieli użyć `renice`. Jak wspomniałem wcześniej, `nice` służy do uruchomienia nowego procesu z określonym priorytetem, a `renice` służy do zmiany priorytetu wcześniej uruchomionego procesu. W tym przykładzie uruchomiliśmy `vim` i ustawiliśmy jego wartość uprzejmości na `10`, używając jednego polecenia.

Zmiana priorytetu edytora tekstu takiego jak Vim może wydawać się dziwnym wyborem jako przykład i faktycznie tak jest. Ale eksperymentowanie z priorytetem procesu edytora Vim jest nieszkodliwe, ponieważ prawdopodobieństwo, że zmiana jego priorytetu doprowadzi do zatrzymania systemu, jest bardzo niewielkie. Nie przychodzi mi do głowy żadne praktyczne, użyteczne zastosowanie zmiany priorytetu czegoś takiego jak edytor tekstu. Najważniejsze jest jednak to, że *możesz* zmienić priorytet procesów działających na Twoim serwerze. Na prawdziwym serwerze możesz mieć ważny proces, który działa i generuje raport, a ten raport musi być dostarczony na czas. A może masz proces, który odpowiada za eksport danych, które muszą na czas dotrzeć do klienta. Tak więc, jeśli pomyślisz o tym szerzej, możesz zastąpić `vim` nazwą procesu, który jest rzeczywiście istotny dla Ciebie lub Twojej organizacji.

Być może zastanawiasz się, co oznacza *nice* w kontekście poleceń `nice` i `renice`. Liczba *nice* zasadniczo odnosi się do tego, na ile uprzejmy jest proces dla innych użytkowników. Im wyższa wartość *nice*, tym niższy priorytet. Tak więc proces o wartości uprzejmości 20 będzie bardziej uprzejmy niż ten o wartości 10. W tym przypadku procesy o wartości 20 mają niższy priorytet, a więc są bardziej uprzejme dla innych procesów w systemie. Parametr uprzejmość może przyjmować wartości z zakresu od –20 do 19. Proces o wartości uprzejmości –20 ma najwyższy możliwy priorytet, natomiast 19 to najniższy priorytet, jaki może przyjąć. Cały system jest bardziej skomplikowany niż ten prosty opis. Chociaż określam wartość uprzejmości jako priorytet procesu, w rzeczywistości nim nie jest. Wartość uprzejmości jest używana do obliczania rzeczywistego priorytetu. Ale na razie, jeśli uprościmy sprawę, przyjmując, że uprzejmość reprezentuje priorytet, przy czym niższa wartość uprzejmości oznacza wyższy priorytet, a wyższa wartość — niższy, będzie to dla nas wystarczające.

Jak dotąd używaliśmy poleceń `nice` i `renice` wraz z opcją `-n` do bezpośredniego ustawiania wartości `nice`. Interesujące może być jednak to, że można uprościć polecenie `renice` i pominąć opcję `-n`:

```
renice 10 42467
```

To polecenie ustawia wartość uprzejmości procesu na +10, podobnie jak w innych naszych przykładach. Możemy również użyć liczby ujemnej dla uprzejmości, jeśli chcemy zwiększyć priorytet procesu:

```
sudo renice -10 42467
```

Chociaż pominięcie opcji `-n` nie oszczędza nam zbytnio pisania na klawiaturze, to przynajmniej wiesz, że jest to możliwe. Dodatkowo w tym przykładzie musiałem użyć `sudo`, ponieważ zmniejszam wartość uprzejmości (więcej na ten temat później).

Jeśli chodzi o polecenie `nice`, możemy również pominąć opcję `-n`, ale polecenie działa w tym zakresie nieco inaczej. Nie zadziała następujące polecenie:

```
nice 15 vim
```

Składnia `nice` jest nieco inna, więc podanie jej bezpośrednio jako parametru liczby dodatniej nie zadziała tak jak w przypadku `renice`. Tutaj będziemy musieli dodać z przodu myślnik:

```
nice -15 vim
```

Patrząc na to polecenie, można pomyśleć, że ustawiamy wartość ujemną. W rzeczywistości tak nie jest. Ponieważ składnia jest inna w `nice`, użyta przez nas wartość `-15` daje w rezultacie dodatnią liczbę równą 15. Potrzebowaliśmy myślnika przed wartością, aby poinformować polecenie `nice`, że wartość jest opcją, którą chcemy ustawić. Jeśli faktycznie chcemy użyć wartości ujemnej z `nice`, unikając jednocześnie opcji `-n`, musimy użyć dwóch myślników:

```
nice --10 vim
```

Różnica w składni między tymi dwoma poleceniami (z i bez opcji `-n`) jest moim zdaniem nieco myląca, więc polecam po prostu używanie opcji `-n` z `nice` i `renice`, ponieważ będzie to bardziej jednolite:

```
nice -n 10 vim
sudo nice -n -10 vim
renice -n 10 42467
sudo renice -n -10 42467
```

Przykłady te pokazują zarówno `nice`, jak i `renice` w wersji z użyciem opcji `-n`, gdy do ustawienia są zarówno wartości dodatnie, jak i ujemne parametru. Ponieważ opcja `-n` jest używana w ten sam sposób w obu poleceniach, może być łatwiejsza do zapamiętania niż skupianie się na detalach składni. W przypadku poleceń, które ustawiają wartość ujemną, użyłem `sudo`, ponieważ jak już wspomniałem wcześniej, tylko *root* może zmienić priorytet procesu lub go uruchomić z parametrem uprzejmości poniżej `0`. Jeśli tak czy inaczej spróbujesz to zrobić, otrzymasz błąd („nie można ustawić uprzejmości: odmowa dostępu”):

```
nice: cannot set niceness: Permission denied
```

Ten rodzaj ochrony jest dość ważny, ponieważ możesz mieć takich użytkowników, którym wydaje się, że ich procesy są najważniejsze, więc próbują nadać im priorytet, sięgając nawet po `-19`. W końcu lepiej, aby to administrator systemu podejmował decyzje, które procesy mogą mieć wartość uprzejmości na minusie.

Jako administrator serwerów Ubuntu sam decydujesz o tym, które procesy powinny być uruchomione, w jaki sposób i z jakim priorytetem. Ty odpowiadasz za to, by system działał w odpowiedni sposób, a dostosowywanie priorytetów dla procesów jest częścią konfigurowania ustawień. Niezależnie od wszystkiego polecenia `nice` i `renice` to kolejne narzędzie w Twoim niezbędniku.

Radzenie sobie z nieprawidłowo działającymi procesami

Jeśli chodzi o polecenie `ps`, w tym momencie wiesz już, jak wyświetlić procesy uruchomione na Twoim serwerze, a także jak zawęzić wyjście według łańcucha lub wykorzystania zasobów. Ale co właściwie możesz zrobić z tą wiedzą? Jakkolwiek nie chcielibyśmy się do tego przyznać, czasami procesy uruchomione na serwerze ulegają awarii lub pracują niewłaściwie i trzeba je zrestartować. Jeśli jakiś proces nie chce się normalnie zamknąć, może być konieczne jego zabicie. W tym rozdziale przedstawiamy polecenia `kill` (zabij) i `killall` (zabij wszystko), którymi można to zrobić.

Polecenie `kill` przyjmuje jako argument identyfikator PID i próbuje elegancko zamknąć proces. W typowej sytuacji, gdy trzeba zakończyć proces, który nie chce się sam zamknąć, najpierw używa się polecenia `ps`, aby znaleźć PID winowajcy. Znając już PID, możesz spróbować wydać polecenie `kill` dla tego procesu. Na przykład, jeśli PID `31258` musi zostać zabity, możesz wydać następujące polecenie:

```
sudo kill 31258
```

Jeśli wszystko pójdzie dobrze, proces zostanie zakończony. Możesz go ponownie uruchomić lub zbadać przyczynę niepowodzenia, przeglądając jego logi.

Aby lepiej zrozumieć, co robi polecenie `kill`, najpierw musisz zrozumieć koncepcję **sygnałów** w Linuksie (zwanych także *przerwaniami programowymi* lub *sygnałami przerwań*). Sygnały są używane zarówno przez administratorów, jak i programistów i mogą być wysyłane do procesu przez jądro, inny proces lub ręcznie za pomocą polecenia. Sygnał instruuje proces o zmianie żądania, a w niektórych przypadkach o całkowitym zakończeniu. Przykładem takiego sygnału jest `SIGHUP`, który informuje procesy, że ich terminal sterujący zakończył pracę. Może to nastąpić, gdy masz otwarty emulator terminala, z kilkoma uruchomionymi w nim procesami. Jeśli zamkniesz okno terminala (bez zatrzymywania uruchomionych procesów), zostanie wysłany do nich sygnał `SIGHUP`, który w zasadzie mówi im, że mają zakończyć pracę (zasadniczo oznacza to, że powłoka zakończyła pracę lub się zawiesiła).

Innymi przykładami są `SIGINT` (w przypadku gdy aplikacja działa na pierwszym planie i jest zatrzymywana przez naciśnięcie *Ctrl+C* na klawiaturze) i `SIGTERM`, który wysyłany do procesu, prosi go o eleganckie zakończenie pracy (ang. *terminate cleanly*). Jeszcze innym przykładem jest `SIGKILL`, który zmusza proces do natychmiastowego zakończenia pracy (ang. *terminate uncleanly*). Każdy sygnał jest reprezentowany przez nazwę, a także wartość liczbową, taką jak `15` dla `SIGTERM` i `9` dla `SIGKILL`. Omawianie każdego z sygnałów wykracza poza zakres tego rozdziału (zaawansowane zagadnienia związane z przerwaniami są przydatne głównie dla programistów), ale jeśli jesteś ciekaw, możesz poczytać więcej na ten temat, zaglądając na stronę *man*.

```
man 7 signal
```

Na potrzeby zagadnień omawianych w tym rozdziale najbardziej interesują nas dwa typy sygnałów, są to sygnały SIGTERM(15) i SIGKILL(9). Kiedy chcemy zatrzymać proces, wysyłamy do niego jeden z tych sygnałów, a komenda kill pozwala nam właśnie to zrobić. Domyślnie polecenie kill wysyła sygnał 15 (SIGTERM), który mówi procesowi, aby elegancko zakończył działanie. Jeśli się to uda, proces zwolni swoją pamięć i zakończy pracę. W naszym poprzednim przykładzie polecenia kill wysłaliśmy do procesu sygnał 15, ponieważ nie sprecyzowaliśmy, jaki sygnał wysłać.

Zakończenie procesu za pomocą SIGKILL(9) jest uważane za skrajną ostateczność. Gdy wysyłasz sygnał 9 do procesu, jest to odpowiednik wyrywania spod niego dywanu lub wysadzania za pomocą laski dynamitu. Proces zostanie zamknięty na siłę, nie będzie miał możliwości i czasu na żadną reakcję, jest to więc jedna z tych rzeczy, których powinieneś unikać, chyba że wcześniej spróbowałeś już wszystkiego, co tylko przyszło Ci do głowy. Teoretycznie wysłanie sygnału 9 może spowodować uszkodzenie plików, problemy z pamięcią lub inne nieprzewidziane skutki. Jeśli chodzi o mnie, nigdy nie natknąłem się na długotrwałe uszkodzenie oprogramowania po użyciu tego sygnału, ale teoretycznie może się tak zdarzyć, więc zapewne będziesz chciał go użyć tylko w ekstremalnych przypadkach. Wysłanie takiego sygnału może być konieczne w przypadku źle działającego procesu (ze statusem *defunct*) lub **procesu zombie**, gdy nie chcą się one same zamknąć. Te procesy są w zasadzie już martwe i zazwyczaj czekają na to, by zostać zamknięte przez swoje procesy macierzyste.

Jeśli proces macierzysty nigdy nie spróbuje takiego procesu zamknąć, pozostanie on na liście procesów. Samo w sobie może i nie jest to wielkim problemem, ponieważ technicznie procesy te i tak nic nie robią. Ale jeśli ich obecność powoduje problemy i nie możesz ich zabić, możesz spróbować wysłać do nich sygnał SIGKILL. Nie powinno być nic złego w wyeliminowaniu procesu zombie, ale najpierw powinieneś dać mu szansę zakończyć działanie w normalny sposób.

Aby wysłać sygnał 9 do procesu, użyj opcji -9 w poleceniu kill. Nie muszę chyba dodawać, że powinieneś się upewnić, że wykonujesz je wobec procesu o właściwym ID:

```
sudo kill -9 31258
```

Po takim poleceniu proces o PID 31258 po prostu zniknie bez śladu. Wszystko, czym się zajmował, zostanie natychmiast usunięte z pamięci. Jeśli z jakiegoś powodu proces nadal będzie działał (co zdarza się niezwykle rzadko), prawdopodobnie będziesz musiał zrestartować serwer, aby się go pozbyć — widziałem taką sytuację tylko w kilku bardzo rzadkich przypadkach. Przykładem takiego procesu jest proces zombie, czyli proces, który pojawia się na liście procesów, ale wysyłanie sygnałów nie ma na niego wpływu, ponieważ i tak nie zostanie mu przydzielony czas procesora. Jeśli kill -9 nie spowoduje pozbycia się procesu, to nic innego tego nie zrobi.

Inną metodą zabicia procesu jest polecenie `killall`, które jest prawdopodobnie bezpieczniejsze niż polecenie `kill` (choćby z tego powodu, że jest mniejsza szansa na przypadkowe zabicie niewłaściwego procesu). Podobnie jak `kill`, `killall` pozwala wysłać `SIGTERM` do procesu, ale w przeciwieństwie do `kill`, można to zrobić, podając nazwę. Ponadto `killall` nie zabija tylko jednego procesu; zabija każdy proces pasujący do (podanej jako opcja) nazwy, który znajdzie. Aby użyć `killall`, po prostu wywołujesz `killall` wraz z nazwą procesu:

```
sudo killall myprocess
```

Podobnie jak w przypadku polecenia `kill`, możesz również wysłać do procesu sygnał `9`:

```
sudo killall -9 myprocess
```

Przypomnę ponownie — używaj tego polecenia tylko w razie potrzeby. W praktyce prawdopodobnie nie będziesz bardzo często używał `killall -9` (jeśli w ogóle będzie Ci potrzebne), ponieważ rzadko zdarza się, aby wiele procesów pod tą samą nazwą przestało działać. Jeśli już musisz wysłać sygnał `9`, to używaj, jeśli tylko możesz, polecenia `kill`.

Polecenia `kill` i `killall` mogą być niezwykle przydatne w sytuacji, gdy proces przestanie działać, ale są to polecenia, których — miejmy nadzieję — nie będziesz musiał używać zbyt często. Takie procesy mogą pojawić się w przypadkach, gdy aplikacje mają do czynienia z sytuacją, z którą nie mogą sobie poradzić. Więc jeśli okaże się, że ciągle musisz zabijać procesy, dobrym pomysłem może być albo sprawdzenie dostępności aktualizacji pakietu odpowiedzialnego za usługę, albo sprawdzenie serwera pod kątem problemów sprzętowych.

W następnym rozdziale przyjrzymy się działającym w tle procesom systemowym świadczącym usługi dla nas lub naszych użytkowników, takim jak proces serwera WWW lub serwera DHCP.

Zarządzanie procesami systemowymi

Procesy systemowe, znane również jako **demony** (ang. *daemons*), to programy, które działają w tle na serwerze i zazwyczaj są automatycznie uruchamiane podczas startu serwera. Zazwyczaj nie zarządzamy nimi bezpośrednio, ponieważ działają one w tle i wykonują swoje zadania, z naszym udziałem lub bez niego. Na przykład, jeśli nasz serwer jest serwerem DHCP i uruchamia proces `isc-dhcp-server`, proces ten będzie działał w tle, nasłuchując żądań DHCP i obsługując żądania przydzielania adresów IP, w miarę jak są zgłaszane. W większości przypadków, gdy instalujemy aplikację, która działa jako usługa, Ubuntu skonfiguruje ją tak, by uruchamiała się podczas uruchamiania serwera, więc nie musimy uruchamiać jej samodzielnie. Zakładając, że usługa

nie napotka problemu, będzie wykonywać swoją pracę, dopóki nie zdecydujemy, by ją zatrzymać. W systemie Linux usługi są zarządzane przez system inicjujący, zwany również PID 1, ponieważ system ten w Linuksie zawsze otrzymuje taki identyfikator. W ostatnich latach sposób zarządzania procesami w serwerze Ubuntu znacznie się zmienił. Obecnie Ubuntu używa `systemd` dla swojego systemu `init`, a jeszcze kilka lat temu był to Upstart. Ubuntu 16.04 było pierwszym wydaniem Ubuntu o długim okresie wsparcia mającym `systemd`, i jest on nadal jest używany w Ubuntu 22.04. Ponieważ `systemd` jest już od dłuższego czasu standardem, skupimy się na związanych z nim poleceniach, używanych do zarządzania naszymi usługami. Starsze systemy `init` są już „na wymarciu".

W `systemd` usługi są znane jako **jednostki** (ang. *units*), ale w większości przypadków terminy „usługa", „demon" i „jednostka" oznaczają w zasadzie to samo. Ponieważ zacząłem używać Linuksa ponad 20 lat temu, nadal z przyzwyczajenia używam nazwy „usługa" dla jednostek `systemd`. Aby pomóc nam zarządzać tymi „jednostkami", `systemd` obsługuje polecenie `systemctl`, dzięki któremu możemy uruchamiać, zatrzymywać i przeglądać status [stan — przyp. tłum.] jednostek na serwerze. Aby pomóc to zilustrować, użyję jako przykładu OpenSSH. Nazwa jednostki jest bez znaczenia, ponieważ składnia polecenia `systemctl` jest taka sama niezależnie od nazwy jednostki, z którą wchodzimy w interakcję. Możesz użyć `systemctl` do uruchomienia, zatrzymania lub zrestartowania instancji serwera Apache, serwera bazy danych, a nawet użyć go do zrestartowania całego stosu sieciowego (ang. *networking stack*). Polecenie `systemctl`, bez żadnych opcji czy parametrów, domyślnie uruchamia się z opcją `list-units`, która zrzuca listę jednostek do powłoki. Może tam być trochę namieszane, ale jeśli znasz nazwę jednostki, którą chcesz znaleźć, możesz przekierować strumień wyjścia do `grep` i szukać konkretnego łańcucha. Przydaje się to również w sytuacji, gdy znasz część nazwy jednostki, a nie jej dokładną nazwę:

```
systemctl | grep ssh
```

Jeśli chcesz sprawdzić stan urządzenia, najlepszym sposobem jest użycie słowa kluczowego `status`, wtedy zobaczysz kilka bardzo użytecznych informacji dotyczących urządzenia. Są to informacje: czy urządzenie jest uruchomione, czy jest włączone (co oznacza, że jest skonfigurowane do uruchamiania podczas startu systemu), jak również ostatnie wpisy w dzienniku dla urządzenia:

```
systemctl status ssh
```

To polecenie da w efekcie coś w rodzaju informacji przedstawionych na rysunku 7.9.

W większości przypadków możesz sprawdzić stan jednostek bez konieczności posiadania dostępu do konta *root*, ale wtedy możesz nie mieć wglądu do wszystkich dostępnych informacji. Na zrzucie ekranu widać kilka wpisów w dzienniku dla usługi `ssh`, ale niektóre jednostki nie pokazują tych wpisów bez `sudo`. Jeśli chodzi o jednostkę `ssh`, wpisy dziennika widzimy podczas sprawdzania stanu zarówno z, jak i bez `sudo`.

Rysunek 7.9. Sprawdzanie stanu urządzenia za pomocą systemctl

Tym, co możesz jeszcze zauważyć na zrzucie ekranu, jest to, że nazwa jednostki `ssh` to w rzeczywistości `ssh.service`, ale nie musisz dołączać części `.service` do nazwy, ponieważ jest to przyjmowane domyślnie. Czasami podczas przeglądania stanu procesu za pomocą `systemctl` wyjście może być skondensowane, aby zaoszczędzić miejsce na ekranie. Aby tego uniknąć i zobaczyć pełne wpisy dziennika, dodaj opcję `-l`:

```
systemctl status -l ssh
```

Inną rzeczą, na którą powinieneś zwrócić uwagę, jest sekcja `vendor preset` jednostki. Większość pakietów w Ubuntu, które zawierają plik serwisowy dla `systemd`, włącza go automatycznie, ale inne dystrybucje zazwyczaj tego nie robią — nie uruchamiają i nie włączają domyślnie jednostek (np. CentOS). W naszym przypadku, gdy chodzi o `ssh`, możesz zobaczyć, że `vendor preset` jest ustawiony na `enabled` (włączony). Oznacza to, że po zainstalowaniu pakietu `openssh-server` jednostka `ssh.service` zostanie automatycznie włączona. Możesz to potwierdzić, sprawdzając linię `Active`, która mówi nam, że jednostka jest uruchomiona (`active (running)`). Linia `Loaded` wyjaśnia, że jednostka jest `enabled` (włączona), więc wiemy, że następnym razem, gdy uruchomimy serwer, `ssh` zostanie automatycznie uruchomiony. Chociaż zazwyczaj w Ubuntu jednostki `systemd` są włączane i uruchamiane automatycznie podczas ich instalacji z pakietu, to nadal może być różnie. Kiedy instalujesz nowy pakiet, sprawdź stan jednostki, by być świadomym jej ustawień.

Uruchamianie i zatrzymywanie jednostki jest równie proste; wszystko, co musisz zrobić, to użyć z `systemctl` odpowiedniego słowa kluczowego: `start` lub `stop` — w zależności od oczekiwanego efektu:

```
sudo systemctl stop ssh
sudo systemctl start ssh
```

Mamy dodatkowe słowa kluczowe, takie jak `restart` (który za jednym zamachem wykonuje polecenia z obu poprzednich przykładów), a niektóre jednostki posiadają nawet funkcję `reload`, która pozwala na aktywowanie nowych ustawień konfiguracyjnych bez konieczności zamykania całej aplikacji. Przykładem, kiedy jest to przydatne, jest Apache, który jest serwerem udostępniającym strony internetowe dla lokalnych lub zewnętrznych użytkowników. Jeśli zatrzymasz serwer Apache, wszyscy użytkownicy zostaną odłączeni od udostępnianej przez niego witryny. Jeśli dodajesz nową stronę, możesz użyć `reload` zamiast `restart`, a wtedy uaktywnisz nową konfigurację, którą dodałeś, bez wpływu na już istniejące połączenia. Serwerowi Apache przyjrzymy się w rozdziale 14., „Udostępnianie serwisów internetowych", więc nie przejmuj się nim zbytnio w tej chwili. Jest to po prostu dobry przykład jednostki, która korzysta z tej dodatkowej funkcjonalności. Ponieważ nie wszystkie jednostki umożliwiają wykorzystanie opcji `reload`, więc dla pewności powinieneś sprawdzić dokumentację aplikacji, która udostępnia daną jednostkę.

Ponieważ w poprzednich przykładach wspomniałem o uruchamianiu i zatrzymywaniu jednostki OpenSSH, ciekawostką jest to, że te działania nie powodują rozłączenia bieżących otwartych sesji SSH. Jeśli zatrzymasz usługę `ssh`, połączenia nie zostaną przerwane. Otwarte połączenia będą utrzymywane, a zatrzymanie SSH tylko zapobiegnie nawiązywaniu nowych połączeń. Dlatego SSH różni się od innych jednostek (takich jak Apache) tym, że istniejące połączenia nie są wyłączane podczas restartu jednostki.

Jak już wspomniałem, jeśli chcesz, aby jednostka była automatycznie uruchamiana podczas startu serwera, musi być włączona (`enabled`). W większości przypadków jednostki są włączane automatycznie, ale jeśli znajdziesz taką, która nie jest, możesz ją włączyć za pomocą słowa kluczowego `enable`:

```
sudo systemctl enable ssh
```

Równie łatwo jest wyłączyć jednostkę:

```
sudo systemctl disable ssh
```

Możesz jednocześnie włączyć jednostkę i ją uruchomić:

```
sudo systemctl enable --now ssh
```

Argument `--now` informuje `systemctl`, by uruchomił jednostkę natychmiast po ustawieniu opcji `enabled` bez oczekiwania na restart serwera lub na uruchomienie jej argumentem `start` w osobnym poleceniu.

Nawet jeśli `systemd` jest używany głównie do zarządzania jednostkami, to w rzeczywistości jest to cała platforma, która zarządza wieloma rzeczami w systemie Linux, w tym rozwiązywaniem DNS, siecią i innymi. `systemd` obsługuje nawet logowanie, jak również zapewnia nam polecenie `journalctl`, którego możemy użyć do przeglądania informacji o logowaniu (jest to również powód, dla którego wyjście `systemctl status ssh` było w stanie pokazać nam wpisy w logu).

Omówiliśmy nieco logowanie w rozdziale 4., „Nawigacja i podstawowe polecenia", a zrobimy to bardziej szczegółowo w rozdziale 22., „Rozwiązywanie problemów z serwerami Ubuntu" (który będzie zawierał również dalsze omówienie polecenia `journalctl`).

Na razie zrozum, że `systemd` jest dość obszerny, jeśli chodzi o liczbę rzeczy, którymi pomaga nam zarządzać. Jeśli jednak rozumiesz procesy uruchamiania, zatrzymywania, włączania, wyłączania i sprawdzania statusu jednostki, to na razie Ci to wystarczy.

Planowanie wykonywania zadań za pomocą polecenia cron

Wcześniej w tym rozdziale omówiliśmy uruchamianie procesów, umożliwianie im działania w tle oraz konfigurowanie uruchomienia wraz ze startem serwera. W niektórych przypadkach może zajść potrzeba wykonania zadania w określonym czasie, a nie ma potrzeby, by działało w tle. W tym miejscu pojawia się `cron`. Za pomocą polecenia `cron` można skonfigurować uruchamianie procesu, programu lub skryptu w określonym czasie z dokładnością do minuty. Każdy użytkownik może mieć swój własny zestaw konfiguracji `cron` (znany jako `crontab`) mogący wykonywać każde zadanie, które mógłby normalnie wykonywać użytkownik. Użytkownik *root* również dysponuje możliwością skonfigurowania `crontab`, z tym że pozwala on na wykonywanie zadań administracyjnych w obrębie całego systemu. Każdy `crontab` zawiera listę zadań `cron` (po jednym w każdej linii), którymi zajmiemy się za chwilę. Aby zobaczyć ustawienia `crontab` dla konta użytkownika, z którego aktualnie korzystasz, możesz użyć polecenia `crontab`:

```
crontab -l
```

Z opcją `-l` polecenie `crontab` pokaże Ci listę zadań dla użytkownika, który uruchomił polecenie. Jeśli wykonasz je jako *root*, zobaczysz `crontab` konta *root*. Jeśli wykonasz je jako użytkownik *jdoe*, zobaczysz `crontab` dla *jdoe*, i tak dalej. Jeśli chcesz zobaczyć `crontab` dla innego użytkownika niż Ty, możesz użyć opcji `-u` i określić użytkownika, ale będziesz musiał wykonać polecenie jako *root* (lub z `sudo`), aby zobaczyć `crontab` dla kogoś innego niż użytkownik, na którego koncie jesteś zalogowany:

```
sudo crontab -u jdoe -l
```

Domyślnie, dopóki nie stworzysz jednego lub więcej zadań, użytkownik nie będzie miał nic ustawione w `crontab`. Dlatego też prawdopodobnie zobaczysz komunikat o braku zadań, gdy sprawdzisz ustawienia dla swoich obecnych użytkowników:

```
no crontab for jdoe
```

Aby utworzyć zadanie cron, najpierw zaloguj się na to konto użytkownika, na którym ma być uruchamiane zadanie. Następnie wydaj polecenie:

```
crontab -e
```

Jeśli w systemie masz zainstalowanych więcej edytorów tekstu niż jeden, możesz zobaczyć wyjście podobne do przedstawionego na rysunku 7.10.

Rysunek 7.10. Wybieranie edytora, który ma być używany z poleceniem crontab

W tym przypadku po prostu naciskasz numer odpowiadający edytorowi tekstu, którego chcesz użyć podczas tworzenia *zadań* w cron. Aby ustawić zmienną środowiskową określającą konkretny edytor i móc edytować crontab za pomocą konkretnego edytora, możesz to zrobić następująco:

```
EDITOR=vim crontab -e
```

W tym przykładzie możesz zastąpić vim dowolnym edytorem tekstu, którego chcesz używać. W tym momencie powinieneś znaleźć się w edytorze tekstu z otwartym plikiem crontab. Domyślny plik crontab dla każdego użytkownika zawiera kilka pomocnych komentarzy, zawierających kilka użytecznych informacji na temat działania cron. Aby dodać nowe zadanie, należy przewinąć plik na sam dół (do miejsca zakończenia wszystkich komentarzy) i wstawić nową linię. Formatowanie wpisu jest tutaj bardzo specyficzne, a przykładowe komentarze w pliku dają Ci wskazówki co do struktury każdej linii. Chodzi konkretnie o tę część:

```
m h dom mon dow command
```

Każde zadanie w cron składa się z sześciu pól, a każde z nich jest oddzielone co najmniej jedną spacją lub tabulatorem. Jeśli użyjesz więcej niż jednej spacji lub tabulatora, cron jest wystarczająco inteligentny, aby poprawnie przetworzyć plik. W pierwszym polu mamy minutę, w której chcielibyśmy, aby zadanie się pojawiło. W drugim polu umieszczamy godzinę w formacie 24-godzinnym, od 0 do 23. Trzecie pole odpowiada za dzień miesiąca. W tym polu można umieścić 5 (5. dzień miesiąca), 23 (23. dzień miesiąca) itd. Czwarte pole odpowiada miesiącowi, np. 3 dla marca lub 12 dla grudnia. Piąte pole to dzień tygodnia, numerowany od 0 do 6, w formacie od niedzieli do soboty. Wreszcie w ostatnim polu znajduje się polecenie, które ma być wykonane. Kilka przykładowych linii crontab wygląda następująco:

```
3 0 * * 4 /usr/local/bin/cleanup.sh
* 0 * * /usr/bin/apt update
0 1 1 * * /usr/local/bin/run_report.sh
```

W pierwszym przykładzie skrypt *cleanup.sh*, znajdujący się w */usr/local/bin*, będzie uruchamiany o godzinie 12:03 w każdy czwartek. Możemy to odczytać, ponieważ kolumna minut ma ustawione `3`, kolumna godzin `0` (północ), kolumna dni `4` (czwartek), a kolumna poleceń zawiera pełną ścieżkę do pliku */usr/local/bin/cleanup.sh*.

Co to znaczy, że polecenie zawiera **pełną ścieżkę**? Generalnie polecenie wpisane w formie pełnej ścieżki oznacza, że jest podana cała ścieżka do plików binarnych odpowiedzialnych za wykonanie polecenia. W drugim przykładzie mogliśmy po prostu wpisać dla polecenia `apt update` i to prawdopodobnie zadziałałoby dobrze. Jednak brak pełnej ścieżki jest uważany za zły zwyczaj w przypadku konfiguracji `cron`. Może tak być, że polecenie zadziała bez podania pełnej ścieżki, ale jest to uzależnione od tego, czy aplikacja zostanie znaleziona w ścieżkach zmiennych środowiskowych użytkownika, który ją wywołuje. Ponieważ różne serwery mogą być skonfigurowane w różny sposób, taka konfiguracja (bez pełnej ścieżki) może zadziałać lub nie — w zależności od tego, jak skonfigurowana jest powłoka. Jeśli wpiszesz pełną ścieżkę, zadanie powinno działać niezależnie od tego, jak skonfigurowana jest powłoka bazowa.

Jeśli nie wiesz, jaka jest pełna ścieżka polecenia, to możesz ją uzyskać, używając polecenia `which`. Polecenie to użyte z nazwą polecenia, które chcesz uruchomić, da Ci pełną ścieżkę dla polecenia, jeśli znajduje się ono w Twoim systemie.

Co do drugiego przykładu: uruchamiamy `/usr/bin/apt update`, aby zaktualizować indeks repozytorium naszego serwera każdego ranka o północy. Gwiazdki w każdym wierszu odnoszą się do dowolnej wartości, więc jeśli w kolumnie minut będą *, oznacza to po prostu, że to zadanie wykonywać się będzie co minutę. W zasadzie jedynym polem, które wymaga objaśnienia, jest pole godziny, które ustawiliśmy na `0`, co reprezentuje godzinę 12:00.

W trzecim przykładzie uruchamiamy skrypt `/usr/local/bin/run_report.sh` pierwszego dnia każdego miesiąca o godzinie 01:00. Jak możesz zauważyć, ustawiliśmy trzecią kolumnę (**dzień w miesiącu**) na `1`, co odpowiada 1 lutego, 1 marca itd. Zadanie to zostanie uruchomione każdego pierwszego dnia miesiąca, ale tylko o godzinie 01:00, ponieważ ustawiłem wartości tylko w pierwszej i drugiej kolumnie, a one reprezentują odpowiednio minutę i godzinę.

Kiedy skończysz edytować plik `crontab` użytkownika i zapiszesz go, `cron` zostanie zaktualizowany i od tego momentu będzie wykonywał zaplanowane zadania w wybranym przez Ciebie czasie. `Crontab` będzie wykonywany zgodnie z aktualnie ustawionym czasem i datą na Twoim serwerze, więc powinieneś się upewnić, że ich ustawienia są poprawne, w przeciwnym razie Twoje zadania wykonają się w innym czasie, niż zakładałeś. Aktualną datę i czas na swoim serwerze możesz sprawdzić, po prostu wydając polecenie `date`.

Najlepszym sposobem na opanowanie tworzenia zadań za pomocą `cron` jest (jak zawsze) praktyka. Drugie przykładowe zadanie `cron` dobrze nadaje się do eksperymentowania, ponieważ aktualizowanie indeksu repozytorium nie wyrządzi żadnej szkody w systemie.

Podsumowanie

W tym rozdziale dowiedziałeś się, jak zarządzać procesami. Zacząłeś od zapoznania się z poleceniem `ps`, którego można użyć do wyświetlenia listy aktualnie uruchomionych procesów. Dowiedziałeś się również trochę o zarządzaniu zadaniami, a także o zabijaniu procesów, które z tego czy innego powodu zachowują się niewłaściwie. Omówiłem również metody zmiany priorytetu dla procesu, co pozwala na uzyskanie pełnej kontroli nad tym, które procesy otrzymują więcej czasu na przetwarzanie danych, a także dowiedziałeś się, w jaki sposób można zaplanować uruchamianie zadań w określonym czasie i dniu za pomocą `cron`.

W rozdziale 8., „Monitorowanie zasobów systemu", dowiesz się, jak można monitorować dostępne na serwerze zasoby. Dotyczy to sprawdzenia wykorzystania dysku, pamięci i przestrzeni wymiany. Zapoznasz się też z kilkoma narzędziami, które mogą sprawić, że zarządzanie zasobami stanie się proste.

Dodatkowe filmy związane z tematem

- Rozpoczęcie pracy z tmux (LearnLinuxTV):
  https://linux.video/tmux-guide
- Procesy działające w tle i na pierwszym planie w systemie Linux (LearnLinuxTV):
  https://linux.video/bg-fg

Lektura uzupełniająca

- Ham Vocke, *A Quick and Easy Guide to tmux*:
  https://learnlinux.link/tmux-article
- Tmux Cheat Sheet & Quick Reference:
  https://learnlinux.link/tcs
- crontab guru:
  https://learnlinux.link/ctg

Skorowidz

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

V

W

Z

Ubuntu Server:
wszystko, czego oczekujesz od najlepszych serwerów!

Ubuntu Server zdobył popularność i uznanie. To zrozumiałe, pozwala bowiem na uzyskanie wysokiej elastyczności i wydajności przy niewielkich kosztach. Od czasu pierwszego wydania w 2004 roku Ubuntu został wzbogacony o potężne i nowoczesne funkcje dla administratorów. Dziś jest najczęściej wdrażaną dystrybucją Linuksa w sieci. System ten jest używany przez organizacje o różnej wielkości i zasobności, w tym przez najbardziej znane korporacje.

To książka przeznaczona dla czytelników o średnich lub zaawansowanych umiejętnościach posługiwania się systemem Linux. Dzięki niej pogłębisz wiedzę o systemach linuksowych i zarządzaniu serwerami działającymi pod kontrolą systemu Ubuntu w rzeczywistych wdrożeniach produkcyjnych. Dowiesz się, od czego zacząć instalację systemu, a następnie jak wprowadzać gotowe rozwiązania w środowisku produkcyjnym. Poznasz w ten sposób narzędzia do administrowania pracą sieci zarówno w małym biurze, jak i w centrum danych. Nauczysz się wdrażać usługi sieciowe, w tym DHCP i DNS. Zobaczysz również, jak skonteneryzować aplikacje za pomocą LXD, aby zmaksymalizować wydajność, i jak budować klastry Kubernetes. W tym wydaniu zaprezentowano Ubuntu w wersji 22.04 LTS, w której zastosowano najnowsze technologie oparte na Linuksie.

Najciekawsze zagadnienia:

- instalacja systemu Ubuntu Server na fizycznych serwerach i na Raspberry Pi
- wdrażanie aplikacji we własnych kontenerach i skalowanie infrastruktury
- automatyzacja wdrożeń i ich konfiguracja
- konteneryzacja aplikacji z wykorzystaniem LXD
- najlepsze praktyki i techniki rozwiązywania problemów

Jay LaCroix jest nauczycielem, inżynierem i ekspertem w dziedzinie serwerów Linuksa. Jest też autorem setek filmów instruktażowych, które zdobyły ogromne uznanie i popularność. W wolnym czasie gra w gry retro i ćwiczy sztuki walki.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-8322-592-0
9 788383 225920
Cena: 129,00 zł
Packt