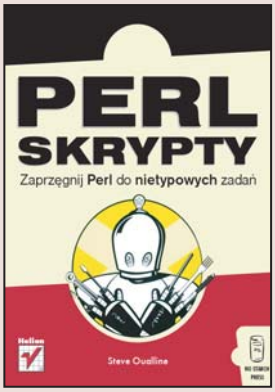
Perl. Skrypty



Autor: Steve Oualline
Tłumaczenie: Rafał Jońca
ISBN: 83-246-0623-8
Tytuł oryginału: Wicked Cool Perl Scripts
Format: B5, stron: 344

Zaprzęgnij Perl do nietypowych zadań

- Zarządzanie witryną WWW
- Administrowanie systemem Linux
- Przetwarzanie grafiki

Chyba każdy programista aplikacji internetowych słyszał o Perlu lub choć raz z niego korzystał. Ten niezwykle popularny język stosowany jest przede wszystkich do tworzenia skryptów CGI będących podstawą dynamicznych witryn WWW. Mimo konkurencji ze strony innych języków skryptowych, język ten nadal święci triumfy, a grono jego użytkowników, skupionych wokół witryny CPAN, stale się powiększa. Perl wykorzystywany jest do różnych zadań, takich jak przetwarzanie plików tekstowych, pobieranie informacji z baz danych, automatyzowanie czynności związanych z publikowaniem treści w witrynach WWW i wiele innych. Jednak nieliczni programiści piszący w Perlu wiedzą, że język ten można wykorzystać również w mniej „typowy" sposób.

W książce „Perl. Skrypty" znajdziesz przykłady takich zastosowań. Dowiesz się, jak użyć Perla do zadań, do których zwykle wykorzystuje się inne technologie. Poznasz sposoby badania spójności witryny WWW, wykrywania ataków hakerskich na serwer i wyszukiwania błędów w skryptach CGI. Nauczysz się wykonywać za pomocą Perla zadania administratora systemów Unix/Linux, pobierać dane z innych serwerów, przetwarzać obrazy i cyfrowe mapy. Zobaczysz także, w jaki sposób można wykorzystać ten język do tworzenia narzędzi wspomagających pracę programisty, nawet takiego, który pisze w innych językach programowania.

- Śledzenie zmian w plikach
- Wykrywanie plików, do których w witrynie WWW nie prowadzą żadne łącza
- Detekcja włamań na serwer
- Usuwanie błędów ze skryptów CGI
- Pobieranie kursów giełdowych
- Zarządzanie użytkownikami w systemie Unix/Linux
- Kontrolowanie procesów w systemie
- Tworzenie internetowych galerii zdjęć
- Narzędzia dla programistów
- Pobieranie map z sieci
- Obsługa wyrażeń regularnych

Ułatw sobie życie i utwórz proste narzędzie rozwiązujące nawet najtrudniejsze problemy

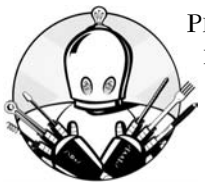
Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

II

6

Administracja systemem Unix



PERL ZOSTAŁ ZAPROJEKTOWANY JAKO JĘZYK PROSTY, BY UŁATWIĆ ADMINISTRATOROM SYSTEMÓW AUTOMATYZACJĘ TYPOWYCH ZADAŃ. IDEALNIE NADAJE SIĘ DO TWORZENIA PROSTYCH SKRYPTÓW ODCIĄŻAJĄCYCH administratorów w niektórych żmudnych zajęciach.

Skrypt 20. Poprawianie błędnych nazw plików

Na początku był wiersz poleceń — nazwy plików były jednolite i oczywiste. Nadeszła jednak era graficznych menedżerów plików i ludzie zaczęli w nazwach plików umieszczać niemalże dowolne znaki. Choć wyglądają ładnie w graficznym interfejsie, stwarzają mnóstwo problemów osobom nadal korzystającym z wiersza poleceń.

Przykładowo muszę radzić sobie z plikami o następujących nazwach:

```
Fibber&Molly [10-1-47] "Fibber's lost $" (v\g snd!).mp3
```

W tej nazwie doliczyłem się 14 znaków, które wymagają specjalnego potraktowania w wierszu poleceń. Jeśli chcę odtworzyć plik z poziomu powłoki, muszę napisać:

```
$ mpg123 Fibber\&Molly\ \[10-1-47\]\ "Fibber\'s lost\ \$"\ \(v\\g\
 snd\!\).mp3
```

Byłoby miło mieć skrypt, który pozbywa się wszystkich trudnych znaków i stosuje dla nich bardziej przyjazne zastępniki. Oto kod takiego skryptu:

Kod

```
 1 #!/usr/bin/perl
 2 foreach my $file_name (@ARGV)
 3 {
 4     # Określ nową nazwę.
 5     my $new_name = $file_name;
 6
 7     $new_name =~ s/[ \t]/_/g;
 8     $new_name =~ s/[\(\)\[\]<>]/x/g;
 9     $new_name =~ s/[\'\`]/=/g;
10     $new_name =~ s/\&/_and_/g;
11     $new_name =~ s/\$/_dol_/g;
12     $new_name =~ s/;/:/g;
13
14     # Upewnij się, że nazwy są różne.
15     if ($file_name ne $new_name)
16     {
17         # Jeśli plik o takiej nazwie już istnieje,
18         # określ nową nazwę.
19         if (-f $new_name)
20         {
21             my $ext = 0;
22
23             while (-f $new_name.".".$ext)
24             {
25                 $ext++;
26             }
27             $new_name = $new_name.".".$ext;
28         }
29         print "$file_name -> $new_name\n";
30         rename($file_name, $new_name);
31     }
32 }
```

Uruchomienie skryptu

Aby uruchomić skrypt, wystarczy jako argumenty przekazać nietypowe nazwy plików.

```
$ fix-names.pl Fibb*
```

(Znaki wieloznaczne doskonale sprawdzają się, gdy trzeba wskazać nietypowe nazwy plików. Przedstawiony wzorzec dopasuje się do podanej wcześniej nazwy pliku).

Wyniki

```
Fibber&Molly [10-1-47] "Fibber's lost $" (v\g snd!).mp3 ->
Fibber_and_Molly_x10-1-47x_"Fibber=s_lost__dol_"_xvxg_snd!x.mp3
```

Jak to działa?

Skrypt analizuje każdy plik przekazany w wierszu poleceń.

```
 2 foreach my $file_name (@ARGV)
```

Następnie określa nową nazwę pliku, zastępując wszystkie nietypowe znaki w nazwie pliku bardziej przyjaznymi odpowiednikami. Pierwsze zastąpienie zamienia wszystkie znaki spacji i tabulacji na znaki podkreślenia. Podkreślenie nie jest spacją, ale wygląda bardzo podobnie.

```
 7     $new_name =~ s/[ \t]/_/g;
```

Podobnie są zastępowane inne nietypowe znaki.

```
 8     $new_name =~ s/[\(\)\[\]<>]/x/g;
 9     $new_name =~ s/[\'\`]/=/g;
10     $new_name =~ s/\&/_and_/g;
11     $new_name =~ s/\$/_dol_/g;
12     $new_name =~ s/;/:/g;
```

Następnie skrypt sprawdza, czy nazwa rzeczywiście się zmieniła. Jeśli nie, nie trzeba wykonywać żadnych dodatkowych działań, ponieważ nazwa jest już przyjazna.

```
14     # Upewnij się, że nazwy są różne.
15     if ($file_name ne $new_name)
16     {
```

Zmiana nazwy nie zadziała, jeśli istnieje inny plik o tej nazwie. Aby uniknąć problemów, skrypt sprawdza ewentualną kolizję nazw i w razie potrzeby odpowiednio modyfikuje wynikową nazwę, dodając numeryczne rozszerzenie do nazwy.

Jeśli zmieniamy nazwę pliku *ten plik* na *ten_plik*, a *ten_plik* już istnieje, skrypt spróbuje zastosować nazwy *ten_plik.1*, *ten_plik.2* itd. — aż do znalezienia wolnej nazwy.

```
17          # Jeśli plik o takiej nazwie już istnieje,
18          # określ nową nazwę.
19          if (-f $new_name)
20          {
21              my $ext = 0;
22
23              while (-f $new_name.".".$ext)
24              {
25                  $ext++;
26              }
27              $new_name = $new_name.".".$ext;
28          }
```

Po wszystkich przekształceniach pozostaje zmienić nazwę pliku.

```
29          print "$file_name -> $new_name\n";
30          rename($file_name, $new_name);
```

Po zmianie nazwy pliku skrypt przechodzi do analizy następnego pliku.

Modyfikacje skryptu

Skrypt nie pozbywa się wszystkich nietypowych znaków. Eliminuje tylko te, z którymi spotkałem się po pobraniu różnych plików z internetu. Rozbudowa o kolejne zamiany znaków jest wyjątkowo prosta. Warto pozostawić w jak największym stopniu oryginalną nazwę pliku, zmieniając znak $ na `_dol_`. Jeśli konieczne są inne odwzorowania, można dowolnie zmodyfikować skrypt.

W trakcie studiów informatycznych założyłem się z jednym kolegą z roku. Moim celem było otworzenie pliku w folderze, którego kolega nie będzie mógł usunąć z własnego komputera. Tworzyłem różne pliki o dziwnych nazwach, na przykład `"skasuj.mnie "` (zauważ spację na końcu), `"-f"` i wiele innych. Niestety, kolega po pewnym czasie usuwał każdy z nich.

W końcu wykorzystałem lukę w systemie plików, by umieścić plik siedem poziomów w głąb struktury plików, choć standardowo system dopuszczał tylko sześć poziomów zagłębienia. System operacyjny nie pozwolił koledze nawet zajrzeć do pliku, nie mówiąc już o jego usunięciu. (Jeśli kogoś to interesuje, systemem operacyjnym był DecSystem-10).

Skrypt 21. Zmiana nazw wielu plików

Standardowe polecenie `rename` systemu Unix lub Linux umożliwia zmianę nazwy tylko jednego pliku w danym momencie (można przenosić folder zawierający wiele plików, ale tak naprawdę zmienia się tylko jedną nazwę). Aby zmienić nazwy wielu plików, potrzebny jest skrypt Perla.

Kod

```
#!/usr/bin/perl
use strict;
use warnings;

use Getopt::Std;
use vars qw/$opt_n $opt_v $opt_e/;

if (not getopts("nve:")) {
    die("Niepoprawne opcje");
}
if (not defined($opt_e)) {
    die("Brak wymaganej opcji -e");
}

foreach my $file_name (@ARGV)
{
    # Określ nową nazwę.
    my $new_name = $file_name;

    # Dokonaj zastąpienia.
    eval "\$new_name =~ s$opt_e";

    # Upewnij się, że nazwy są różne.
    if ($file_name ne $new_name)
    {
        # Jeśli istnieje plik o tej nazwie,
        # określ nową nazwę.
        if (-f $new_name)
        {
            my $ext = 0;

            while (-f $new_name."."$ext)
            {
                $ext++;
            }
            $new_name = $new_name."."$ext;
        }
        if ($opt_v) {
            print "$file_name -> $new_name\n";
        }
```

```
41          if (not defined($opt_n)) {
42              rename($file_name, $new_name);
43          }
44      }
45 }
```

Uruchomienie skryptu

Skrypt przyjmuje następujące parametry:

`-e '/stara/nowa/znaczniki'` — wzorzec podmiany (w wersji używanej w poleceniu `"=~ s..."` Perla).

`-n` — nie dokonuje rzeczywistej zamiany nazw plików.

`-v` — wyświetla informacje o swoim działaniu.

Wszystkie pozostałe parametry to nazwy plików do podmiany. Oto przykład działania:

```
$ mass-rename.pl -e '/\.3/\MP3/' test/D*.3
```

Wyniki

```
test/Dragnet_50_1_14.3 -> test/Dragnet_50_1_14.MP3
test/Dragnet_50_1_21.3 -> test/Dragnet_50_1_21.MP3
test/Dragnet_50_1_7.3 -> test/Dragnet_50_1_7.MP3
```

Jak to działa?

Skrypt rozpoczyna się od przetworzenia wiersza poleceń. W tym celu stosuje moduł `Getopt::Std`.

```
 8 if (not getopts("nve:")) {
 9     die("Niepoprawne opcje");
10 }
```

Opcja `-e` jest wymagana, więc skrypt sprawdza, czy została podana.

```
11 if (not defined($opt_e)) {
12     die("Brak wymaganej opcji -e");
13 }
```

Przetwarza każdy plik.

```
15 foreach my $file_name (@ARGV)
16 {
17     # Określ nową nazwę.
18     my $new_name = $file_name;
```

Stara nazwa zostaje zamieniona na nową przy użyciu operatora `eval`. Funkcja ta traktuje swój argument jako instrukcję języka Perl i ją wykonuje. Sposób korzystania z funkcji nie jest najprostszy.

W tym programie wzorzec zamiany (parametr `-e`) zostaje umieszczony w ciągu znaków. Nowy wynik ma zostać umieszczony w `$new_name`. Jeśli po prostu umieściłoby się ten tekst w programie bez otaczania go cudzysłowami, pojawiłaby się informacja o błędzie. Trzeba również w specjalny sposób potraktować znak `$`, gdyż tradycyjnie powoduje on zastosowanie `$new_name` jako argumentu funkcji `eval`. Ponieważ przypisanie do zmiennej ma nastąpić we wnętrzu funkcji `eval`, musimy znieść specjalne znaczenie znaku dolara.

```
20     # Dokonaj zastąpienia.
21     eval "\$new_name =~ s$opt_e";
```

Po określeniu nowej nazwy problem istnienia pliku o nowej nazwie zostaje rozwiązany w taki sam sposób jak we wcześniejszym skrypcie.

Jeżeli użytkownik przekazał argument `-v`, skrypt wyświetla wszystkie wykonywane operacje. Przy braku argumentu `-n` wykonuje zmianę nazwy pliku.

```
38         if ($opt_v) {
39             print "$file_name -> $new_name\n";
40         }
41         if (not defined($opt_n)) {
42             rename($file_name, $new_name);
43         }
```

Modyfikacje skryptu

Skrypt został zaprojektowany dla osób, które wiedzą, co robią. Z tego powodu brakuje w nim wielu sprawdzeń, które powinny znaleźć się w skrypcie kierowanym do typowego użytkownika. Wyrażenie zmieniające nazwy nie jest weryfikowane. Skrypt nie stosuje trybu interaktywnego z potwierdzaniem każdej zmiany nazwy.

Choć skrypt był projektowany do zmiany nazw plików, przy odrobinie pracy może posłużyć również do masowego modyfikowania dowiązań symbolicznych. Taki skrypt okazuje się wprost nieoceniony przy zmianie dysku, gdy trzeba zmodyfikować wszystkie dowiązania symboliczne związane ze starymi plikami.

Ten skrypt to doskonały przykład, w jaki sposób dobry skrypt Perla potrafi odciążyć administratora w wykonywaniu żmudnych zadań.

Skrypt 22. Sprawdzanie dowiązań symbolicznych

Dowiązania symboliczne są wielce pożyteczne, ale potrafią sprawić mnóstwo problemów, jeśli nie są poprawne. Skrypt sprawdza drzewo folderów w poszukiwaniu dowiązań symbolicznych i upewnia się, że są prawidłowe.

Kod

```
 1 #!/usr/bin/perl
 2 use strict;
 3 use warnings;
 4
 5 use File::Find ();
 6
 7 use vars qw/*name *dir *prune/;
 8 *name   = *File::Find::name;
 9 *dir    = *File::Find::dir;
10 *prune  = *File::Find::prune;
11
12 # Przejście przez system plików.
13 File::Find::find({wanted => \&wanted}, @ARGV);
14 exit;
15
16
17 sub wanted {
18     if (-l $_) {
19         my @stat = stat($_);
20         if ($#stat == -1) {
21             print "Niepoprawne dowiązanie: $name\n";
22         }
23     }
24 }
```

Uruchomienie skryptu

Skrypt jako argument przyjmuje folder lub zbiór folderów. Następnie przechodzi przez drzewo plików zaczynające się w danym folderze, szukając błędnych dowiązań symbolicznych.

```
$ sym-check.pl folder
```

Wyniki

```
Niepoprawne dowiązanie: folder/dowiązanie_donikąd
```

Jak to działa?

Moduł `File::Find` przeszukuje drzewo plików. Funkcja `find` przechodzi przez każdy plik w drzewie folderów i wywołuje dla każdego z nich podprocedurę `wanted`.

```
12 # Przejście przez system plików.
13 File::Find::find({wanted => \&wanted}, @ARGV);
```

Funkcja `wanted` sprawdza najpierw, czy plik jest dowiązaniem symbolicznym, a następnie korzysta z funkcji `stat`. Funkcja `stat` zwraca informacje na temat rzeczywistego pliku, a nie dowiązania symbolicznego (informacje o dowiązaniu zwraca funkcja `lstat`).

Jeżeli dowiązanie symboliczne nie jest poprawne, funkcja `stat` zwróci pustą listę. W takim przypadku wyświetlamy komunikat o błędzie.

```
17 sub wanted {
18     if (-l $_) {
19         my @stat = stat($_);
20         if ($#stat == -1) {
21             print "Niepoprawne dowiązanie: $name\n";
22         }
23     }
24 }
```

Dodatkowa uwaga: zmienna `$_` zawiera nazwę pliku określoną względem aktualnego folderu. Funkcja `find` zmienia aktualny folder, więc `$_` działa dla operatora `-l` i funkcji `stat`, ale nie nadaje się do użycia w celu wyświetlenia komunikatu. W tym celu lepiej zastosować zmienną `$name` zawierającą pełną nazwę pliku.

Modyfikacje skryptu

Skrypt został oryginalnie napisany z wykorzystaniem polecenia `find2perl`. Zmieniłem funkcję `wanted` w taki sposób, by odpowiadała nowemu sposobowi działania skryptu. Moduł `File::Find` pozwala lokalizować różnego rodzaju pliki. Wystarczy wskazać mu, co ma zostać znalezione, a następnie zmienić wygenerowany skrypt.

Ciekawym rozwiązaniem byłby skrypt, który starałby się naprawiać lub usuwać w sposób interaktywny błędne dowiązania symboliczne. Skrypt doskonale znajduje potencjalne źródła problemów. Sposób jego wykorzystania zależy od użytkownika.

Skrypt 23. — Alarmowanie o braku miejsca na dysku twardym

Dzisiaj zabrakło mi miejsca na dysku twardym. Pracowałem nad programem, który powodował powstawanie ogromnych zrzutów pamięci po napotkaniu błędu i w ten sposób wypełnił cały dysk. Oczywiście nie zauważyłem problemu, dopóki w trakcie kompilacji pliki obiektów nie zaczęły być obcinane. Warto byłoby poznać przyczynę powstawania błędu nieco wcześniej. Gdy poznałem przyczynę błędu, byłem zmuszony usunąć wszystkie skompilowane pliki projektu i skompilować je ponownie.

Skrypt informuje wszystkich użytkowników, że zaczyna brakować miejsca na dysku twardym.

Kod

```
 1 #!/usr/bin/perl
 2 use strict;
 3 use warnings;
 4
 5 use Filesys::DiskSpace;
 6
 7 my $space_limit = 5;    # Jeśli mniej niż 5%, informuj...
 8
 9 if ($#ARGV == -1) {
10     print "Użycie: $0 <sp> [<sp>....]\n";
11     exit (8);
12 }
13
14 # Sprawdź każdy folder
15 # na liście.
16 foreach my $dir (@ARGV) {
17     # Pobierz informacje o systemie plików.
18     my ($fs_type, $fs_desc, $used,
19         $avail, $fused, $favail) = df $dir;
20
21     # Ilość wolnego miejsca.
22     my $per_free = (($avail) / ($avail+$used)) * 100.0;
23     if ($per_free < $space_limit) {
24         # Zmodyfikuj komunikat według własnych potrzeb...
25         my $msg = sprintf(
26           "OSTRZEŻENIE: Ilość wolnego miejsca na $dir ".
27               "spadła do poziomu %0.2f%%",
28           $per_free);
29         system("wall '$msg'");
30     }
31 }
```

Uruchomienie skryptu

Do uruchamiania skryptu najlepiej zastosować zadanie harmonogramowane (używając narzędzia `cron`). Aby uruchomić je ręcznie, wystarczy podać nazwy jednego lub kilku folderów w wierszu poleceń.

```
$ disk.pl /home
```

Wyniki

Jeżeli na dysku jest wystarczająca ilość miejsca, nic się nie stanie. W przeciwnym razie wszyscy użytkownicy systemu otrzymają następujący komunikat:

```
Broadcast message from root(pts/6) (Thu Oct 28 20:19:13 2004):

OSTRZEŻENIE: Ilość wolnego miejsca na /home spadła do poziomu 4.00%
```

Jak to działa?

Skrypt przechodzi przez poszczególne foldery podane w wierszu poleceń i sprawdza ilość wolnego miejsca.

```
16 foreach my $dir (@ARGV) {
```

Moduł `Filesys::DiskSpace` informuje o ilości miejsca zajętego na dysku twardym. W ten sposób łatwo wyliczyć ilość wolnego miejsca w procentach.

```
17     # Pobierz informacje o systemie plików.
18     my ($fs_type, $fs_desc, $used,
19         $avail, $fused, $favail) = df $dir;
20
21     # Ilość wolnego miejsca.
22     my $per_free = (($avail) / ($avail+$used)) * 100.0;
```

Skrypt sprawdza, czy obliczona wartość znajduje się poniżej wskazanego limitu.

```
23     if ($per_free < $space_limit) {
24         # Zmodyfikuj komunikat dla własnych potrzeb...
```

Gdy miejsca brakuje, warto użyć polecenia `wall`, aby poinformować o tym fakcie wszystkich użytkowników.

```
25          my $msg = sprintf(
26            "OSTRZEŻENIE: Ilość wolnego miejsca na $dir ".
27                "spadła do poziomu %0.2f%%",
28            $per_free);
29          system("wall '$msg'");
30      }
31  }
```

Modyfikacje skryptu

Graniczna ilość wolnego miejsca jest na stałe zakodowana na wartość 5%. Jeżeli spadnie poniżej tego limitu, pojawi się ostrzeżenie. Wartość limitu można łatwo zmienić, dostosowując ją do własnych potrzeb.

Skrypt jedynie ostrzega wszystkich użytkowników. Bardziej zaawansowanym rozwiązaniem byłoby usuwanie plików tymczasowych, starych plików dzienników i plików zrzutu pamięci po napotkaniu błędu.

Skrypt doskonale radzi sobie ze znajdowaniem problemu i pozwala szybko sobie z nim poradzić.

Skrypt 24. Dodanie nowego użytkownika

Istnieje wiele programów zapewniających dodawanie nowych użytkowników do systemu Unix lub Linux. Wystarczy wypełnić puste pola, kliknąć przycisk *Dodaj*, i koniec. Dlaczego robić sobie kłopot i pisać nowy skrypt wykonujący to zadanie?

Jeśli dodaje się jednego użytkownika, skrypt jest bezużyteczny. Jeżeli trzeba dodać kilka tysięcy użytkowników, pozwala zaoszczędzić mnóstwo czasu, gdyż zapewnia działanie wsadowe. (Gdy administrator pracuje na uniwersytecie, może zastosować ten skrypt do tworzenia kont nowym studentom na podstawie uzyskanej listy ich nazwisk).

Kod

```
 1 #!/usr/bin/perl
 2 use strict;
 3 use warnings;
 4 use Fcntl ':flock'; # Importuje stałe LOCK_*.
 5
 6 #  Modyfikowane pliki.
 7 my $pw_file = "/etc/passwd";
 8 my $group_file = "/etc/group";
 9 my $shadow_file = "/etc/shadow";
10
```

```
11 # Pobierz nazwę użytkownika.
12 my $login;         # Nazwa użytkownika.
13 print "Nazwa użytkownika: ";
14 $login = <STDIN>;
15 chomp($login);
16
17 if ($login !~ /[A-Z_a-z0-9]+/) {
18     die("Nie podano poprawnej nazwy użytkownika.");
19 }
20
21 open PW_FILE, "<$pw_file" or die("Nieudany odczyt pliku $pw_file");
22 # Zablokuj plik na czas trwania skryptu.
23 flock PW_FILE, LOCK_EX;
24
25 # Sprawdź informacje na temat nazwy użytkownika.
26 my $check_uid = getpwnam($login);
27 if (defined($check_uid)) {
28     print "$login już istnieje\n";
29     exit (8);
30 }
31
32 # Znajdź najwyższy UID.  Nowy będzie większy o 1.
33 my @pw_info = <PW_FILE>;
34
35 my $uid = 0;       # UID dla użytkownika.
36
37 # Znajdź największego użytkownika.
38 foreach my $cur_pw (@pw_info) {
39     my @fields = split /:/, $cur_pw;
40     if ($fields[2] > 60000) {
41         next;
42     }
43     if ($fields[2] > $uid) {
44         $uid = $fields[2];
45     }
46 }
47 $uid++;
48
49 # Każdy użytkownik otrzymuje własną grupę.
50 my $gid = $uid;
51
52 # Domyślny folder macierzysty.
53 my $home_dir = "/home/$login";
54
55 print "Imię i nazwisko: ";
56 my $full_name = <STDIN>;
57 chomp($full_name);
58
```

```
59 my $shell = "";    # Stosowana powłoka.
60 while (! -f $shell) {
61     print "Powłoka: ";
62     $shell = <STDIN>;
63     chomp($shell);
64 }
65
66 print "Tworzę konto dla $login [$full_name]\n";
67
68 open PW_FILE, ">>$pw_file" or
69     die("Nieudane dodanie informacji do $pw_file");
70 print PW_FILE
71 "${login}:x:${uid}:${gid}:${full_name}:${home_dir}:$shell\n";
72
73 open GROUP_FILE, ">>$group_file" or
74    die("Nieudane dodanie informacji do $group_file");
75 print GROUP_FILE "${login}:x:${gid}:$login\n";
76 close GROUP_FILE;
77
78 open SHADOW, ">>$shadow_file" or
79     die("Nieudane dodanie informacji do $shadow_file");
80 print SHADOW "${login}:*:11647:0:99999:7:::\n";
81 close SHADOW;
82
83 # Utwórz folder macierzysty i wypełnij go niezbędnymi plikami.
84 mkdir($home_dir);
85 chmod(0755, $home_dir);
86 system("cp -R /etc/skel/.[a-zA-Z]* $home_dir");
87 system("find $home_dir -print ".
88        "-exec chown ${login}:${login} {} \\;");
89
90 # Ustaw hasło dla użytkownika.
91 print "Ustawiam hasło\n";
92 system("passwd $login");
93
94 flock(PW_FILE,LOCK_UN);
95 close(PW_FILE);
```

Uruchomienie skryptu

Skrypt jest interaktywny. Uruchamia się go bez parametrów. Skrypt prosi o bezpośrednie wpisanie wszystkich danych.

UWAGA *Skrypt jest silnie powiązany z konkretnym systemem i potrafi uszkodzić system o innej strukturze. Należy utworzyć kopię bezpieczeństwa krytycznych plików, sprawdzić poprawność skryptu dla danego systemu i przetestować go na mniej istotnym komputerze.*

Wyniki

```
# add_user.pl
Nazwa użytkownika: jeffar
Imię i nazwisko: Rafał Jońca
Powłoka: /bin/bash
Tworzę konto dla jeffar [Rafał Jońca]
/home/jeffar
/home/jeffar/.bash_logout
/home/jeffar/.bash_profile
/home/jeffar/.bashrc
/home/jeffar/.mailcap
/home/jeffar/.screenrc
Ustawiam hasło
Zmiana hasła dla użytkownika jeffar.
Nowe hasło UNIX:
Powtórz nowe hasło UNIX:
passwd: wszystkie tokeny uwierzytelniające zostały poprawnie uaktualnione.
```

Jak to działa?

W zasadzie tworzenie nowego użytkownika nie jest trudnym procesem. Należy zmodyfikować kilka plików. Z drugiej strony pomyłka może mieć bardzo nieciekawe skutki, łącznie z uniemożliwieniem dostępu komukolwiek do komputera (brak możliwości zalogowania się).

Skrypt wykonuje następujące kroki:

1. Pobiera nazwę użytkownika od operatora.
2. Blokuje plik haseł.
3. Sprawdza, czy użytkownik rzeczywiście nie istnieje.
4. Generuje identyfikator użytkownika (UID).
5. Tworzy wpis w pliku */etc/passwd*.
6. Tworzy wpis w pliku */etc/groups*.
7. Tworzy wpis w pliku */etc/shadow*.
8. Tworzy folder macierzysty użytkownika.
9. Kopiuje wszystkie pliki z folderu przykładowego szkieletu (*/etc/skel*) do nowego folderu macierzystego.
10. Zmienia właściciela wszystkich skopiowanych plików na nowego użytkownika.
11. Wywołuje program `passwd`, który zapewnia wygenerowanie nowego hasła dla użytkownika.
12. Odblokowuje plik */etc/passwd*.

Każdy z kroków jest prosty, ale zapamiętanie ich wszystkich bywa problematyczne.

Prześledźmy, w jaki sposób skrypt wykonuje każdy z kroków.

1. Pobiera nazwę użytkownika od operatora. Dokonuje dodatkowo walidacji, by sprawdzić, czy nazwa użytkownika jest zgodna z przyjętymi zasadami.

```
11 # Pobierz nazwę użytkownika.
12 my $login;    # Nazwa użytkownika.
13 print "Nazwa użytkownika: ";
14 $login = <STDIN>;
15 chomp($login);
16
17 if ($login !~ /[A-Z_a-z0-9]+/) {
18     die("Nie podano poprawnej nazwy użytkownika.");
19 }
```

2. Blokuje plik haseł. W ten sposób skrypt zabezpiecza się przed modyfikacją pliku przez inną osobę w trakcie dodawania użytkownika.

```
21 open PW_FILE, "<$pw_file" or die("Nieudany odczyt pliku $pw_file");
22 # Zablokuj plik na czas trwania skryptu.
23 flock PW_FILE, LOCK_EX;
```

3. Sprawdza, czy użytkownik rzeczywiście nie istnieje. Zadanie to można wykonać, pobierając UID nowego użytkownika. Ponieważ użytkownik nie istnieje, polecenie nie zadziała i zwróci niezdefiniowaną wartość.

```
25 # Sprawdź informacje na temat nazwy użytkownika.
26 my $check_uid = getpwnam($login);
27 if (defined($check_uid)) {
28     print "$login już istnieje\n";
29     exit (8);
30 }
```

4. Generuje identyfikator użytkownika (UID). Program analizuje plik haseł i znajduje najwyższy UID, który jest mniejszy od 60000. Limit ten wynika z faktu, iż istnieją specjalne UID o wyższych numerach. Przykładowo konto nobody ma UID o numerze 65534.

   Identyfikator nowego użytkownika będzie o 1 wyższy od znalezionego UID.

```
32 # Znajdź najwyższy UID. Nowy będzie większy o 1.
33 my @pw_info = <PW_FILE>;
34
35 my $uid = 0;    # UID dla użytkownika.
36
37 # Znajdź największego użytkownika.
38 foreach my $cur_pw (@pw_info) {
39     my @fields = split /:/, $cur_pw;
40     if ($fields[2] > 60000) {
41         next;
42     }
43     if ($fields[2] > $uid) {
44         $uid = $fields[2];
45     }
46 }
47 $uid++;
```

5. Skrypt pobiera dodatkowe informacje wymagane do utworzenia wpisu w pliku haseł. Zakłada również, że GUI = UID, czyli że każdy użytkownik ma własną grupę. Po uzyskaniu tych informacji można utworzyć wpis w pliku */etc/passwd*.

```
68 open PW_FILE, ">>$pw_file" or
69     die("Nieudane dodanie informacji do $pw_file");
70 print PW_FILE
71 "${login}:x:${uid}:${gid}:${full_name}:${home_dir}:$shell\n";
```

6. Tworzy wpis w pliku */etc/groups*.

```
73 open GROUP_FILE, ">>$group_file" or
74    die("Nieudane dodanie informacji do $group_file");
75 print GROUP_FILE "${login}:x:${gid}:$login\n";
76 close GROUP_FILE;
```

7. Tworzy wpis w pliku */etc/shadow*.

```
78 open SHADOW, ">>$shadow_file" or
79     die("Nieudane dodanie informacji do $shadow_file");
80 print SHADOW "${login}:*:11647:0:99999:7:::\n";
81 close SHADOW;
```

8. Tworzy folder macierzysty użytkownika.

```
83 # Utwórz folder macierzysty i wypełnij go niezbędnymi plikami.
84 mkdir($home_dir);
85 chmod(0755, $home_dir);
```

9. Kopiuje wszystkie pliki z folderu przykładowego szkieletu (*/etc/skel*) do nowego folderu macierzystego.

```
86 system("cp -R /etc/skel/.[a-zA-Z]* $home_dir");
```

10. Zmienia właściciela wszystkich skopiowanych plików na nowego użytkownika.

```
87 system("find $home_dir -print ".
88        "-exec chown ${login}:${login} {} \\;");
```

11. Wywołuje program `passwd`, który zapewnia wygenerowanie nowego hasła dla użytkownika.

```
90 # Ustaw hasło dla użytkownika.
91 print "Ustawiam hasło\n";
92 system("passwd $login");
```

12. Odblokowuje plik */etc/passwd*.

```
94 flock(PW_FILE,LOCK_UN);
```

Modyfikacje skryptu

Skrypt pobiera nazwę użytkownika i inne informacje, stosując interaktywne zapytania. Nic nie stoi na przeszkodzie, by móc te same informacje pobierać z pliku konfiguracyjnego lub nawet listy studentów uzyskanej z dziekanatu. Skrypt wykonuje swe zadanie niezależnie od tego, skąd pochodzą dane.

Skrypt 25. Zablokowanie użytkownika

Gdy jeden ze studentów nie zachowywał się poprawnie i próbował złamać system, można na kilka tygodni zablokować mu dostęp do konta.

UWAGA *Skrypt jest dostosowany do konkretnego rodzaju systemu operacyjnego. Przed jego uruchomieniem warto sprawdzić, czy będzie działał poprawnie również na innym systemie.*

Kod

```
 1 #!/usr/bin/perl
 2 use strict;
 3 use warnings;
 4
 5 if ($#ARGV != 0) {
 6     print STDERR "Użycie: $0 <konto>\n";
 7 }
 8
 9 my $user = $ARGV[0];
10
11 # Pobierz informacje związane z logowaniem.
12 my $uid = getpwnam($user);
13 if (not defined($uid)) {
14     print "$user nie istnieje.\n";
15     exit (8);
16 }
17
18 system("passwd -l $user");
19 my @who = `who`;
20 @who = grep /^$user\s/,@who;
21 foreach my $cur_who (@who) {
22     my @words = split /\s+/, $cur_who;
23     my $tty = $words[1];
24
25     if (not open(YELL, ">>/dev/$tty")) {
26         next;
27     }
28     print YELL <<EOF ;
29 ************************************************************
30 PILNA UWAGA OD ADMINISTRATORA SYSTEMU
31
32 To konto zostało zawieszone. Za 10 sekund zostaniesz
33 odłączony od systemu. Dokonaj natychmiastowego wylogowania.
34 ************************************************************
35 EOF
36     close YELL;
37 }
38 sleep(10);
39 my @procs = `ps -u $user`;
40 shift @procs;
41 foreach my $cur_proc (@procs) {
42     $cur_proc =~ /(\d+)/;
43     if (defined($1)) {
44         print "Zabijam $1\n";
45         kill 9, $1;
46     }
47 }
```

Uruchomienie skryptu

Skrypt przyjmuje jeden parametr: nazwę użytkownika.

```
# dis_user.pl jeffar
```

Wyniki

```
Blokowanie hasła dla użytkownika jeffar
passwd: wykonane
```

Jeśli użytkownik jest aktualnie zalogowany, z pewnością będzie zszokowany. Wiadomość pojawi się na jego terminalu.

```
************************************************************
PILNA UWAGA OD ADMINISTRATORA SYSTEMU

To konto zostało zawieszone. Za 10 sekund zostaniesz
odłączony od systemu. Dokonaj natychmiastowego wylogowania.
************************************************************
```

Po 10 sekundach użytkownik jest automatycznie wylogowywany, czy tego chce, czy nie.

Jak to działa?

Skrypt sprawdza najpierw, czy użytkownik istnieje, używając metody `getpwnam`, stosowanej również w poprzednim skrypcie.

Następnie wywołuje program `passwd`, by zablokować użytkownika.

```
18 system("passwd -l $user");
```

W następnym kroku używa polecenia `who`, by sprawdzić, czy użytkownik jest zalogowany. Jeśli tak, określa terminal, na którym aktualnie pracuje.

```
19 my @who = `who`;
20 @who = grep /^$user\s/,@who;
21 foreach my $cur_who (@who) {
22     my @words = split /\s+/, $cur_who;
23     my $tty = $words[1];
```

Otwiera terminal i informuje użytkownika o blokowaniu konta za pomocą rzucającego się w oczy komunikatu.

```
25     if (not open(YELL, ">>/dev/$tty")) {
26         next;
27     }
28     print YELL <<EOF ;
29 ************************************************************
30 PILNA UWAGA OD ADMINISTRATORA SYSTEMU
31
32 To konto zostało zawieszone. Za 10 sekund zostaniesz
33 odłączony od systemu. Dokonaj natychmiastowego wylogowania.
34 ************************************************************
35 EOF
36     close YELL;
37 }
```

Ponieważ użytkownik ma 10 sekund na reakcję, skrypt czeka przez ten okres.

```
38 sleep(10);
```

Polecenie `ps` pozwala pobrać wszystkie procesy należące do użytkownika. Pierwszy wiersz wyników należy pominąć, gdyż zawiera nagłówek informacyjny. Skrypt przetwarza pozostałe wiersze.

```
39 my @procs = `ps -u $user`;
40 shift @procs;
```

W trakcie przetwarzania wydobywa z każdego wiersza informację na temat identyfikatora procesu, którego właścicielem jest użytkownik. Zdobyta informacja pozwala wykonać polecenie `kill` dla procesu, wymuszając w ten sposób jego zakończenie.

```
41 foreach my $cur_proc (@procs) {
42     $cur_proc =~ /(\d+)/;
43     if (defined($1)) {
44         print "Zabijam $1\n";
45         kill 9, $1;
46     }
47 }
```

W tym momencie użytkownik został usunięty, a konto — zablokowane.

Modyfikacje skryptu

Skrypt uzależniony jest od wielu poleceń zewnętrznych, na przykład `ps` i `who`. Sposób działania tych poleceń zależy od wykorzystywanego systemu operacyjnego, więc konieczne może okazać się dostosowanie skryptu do własnego systemu.

Skrypt 26. Usunięcie użytkownika

Użytkownik został zablokowany. Następny krok to jego usunięcie.

OSTRZEŻENIE *Ten skrypt może zniszczyć dane systemowe. Jego działanie jest silnie uzależnione od konkretnego systemu operacyjnego. Należy być bardzo ostrożnym przy jego stosowaniu.*

Kod

```
 1 #!/usr/bin/perl
 2 use strict;
 3 use warnings;
 4 use Fcntl ':flock';    # Importuje stałe LOCK_*.
 5
 6 if ($#ARGV != 0) {
 7     print STDERR "użycie: $0 <użytkownik>\n";
 8     exit (8);
 9 }
10
11 my $user = $ARGV[0];
12
13 sub edit_file($)
14 {
15     my $file = shift;
16
17     open IN_FILE, "<$file" or
18         die("Nieudane otwarcie pliku $file do odczytu");
19
20     open OUT_FILE, ">$file.new" or
21         die("Nieudane otwarcie pliku $file.new do zapisu");
22
23     while (1) {
24         my $line = <IN_FILE>;
25         if (not defined($line)) {
26             last;
27         }
28         if ($line =~ /^$user/) {
29             next;
```

```
30          }
31          print OUT_FILE $line;
32      }
33      close (IN_FILE);
34      close (OUT_FILE);
35      unlink("$file.bak");
36      rename("$file", "$file.bak");
37      rename("$file.new", $file);
38  }
39
40  my @info = getpwnam($user);
41  if (@info == -1) {
42      die("Brak użytkownika $user");
43  }
44
45  open PW_FILE, "</etc/passwd" or
46      die("Nieudany odczyt pliku /etc/passwd");
47
48  # Zablokuj plik na czas trwania skryptu.
49  flock PW_FILE, LOCK_EX;
50
51  edit_file("/etc/group");
52  edit_file("/etc/shadow");
53
54  if ($info[7] eq "/home/$user") {
55      system("rm -rf /home/$user");
56  } else {
57      print "Użytkownik wykorzystuje niestandardowy folder macierzysty.\n";
58      print "Usuń go ręcznie.\n";
59      print "Folder = $info[7]\n";
60  }
61  print "Użytkownik $user został usunięty\n";
62
63  edit_file("/etc/passwd");
64
65  flock(PW_FILE,LOCK_UN);
66  close(PW_FILE);
```

Uruchomienie skryptu

Użytkownika do usunięcia podaje się jako argument wiersza poleceń.

```
# del_user.pl jeffar
```

Wyniki

```
# del_user.pl jeffar
Użytkownik jeffar został usunięty
```

Jak to działa?

Skrypt edytuje pliki */etc/group*, */etc/shadow* i */etc/passwd*, by usunąć z nich wszelkie informacje o użytkowniku. Skrypt odczytuje plik wiersz po wierszu i szuka tych, które zawierają nazwę użytkownika i za nim znak dwukropka. Takie wiersze są pomijane przy zapisie nowej wersji pliku.

Funkcja `edit_file` odczytuje dane z pliku (na przykład */etc/group*) i zapisuje plik o tej samej nazwie, ale z rozszerzeniem *.new* (na przykład */etc/group.new*). Po zakończeniu edycji pliku dokonuje następującej zamiany:

```
/etc/group -> /etc/group.bak
/etc/group.new -> /etc/group
```

Skrypt usuwa również folder macierzysty użytkownika, stosując poniższy kod:

```
54 if ($info[7] eq "/home/$user") {
55     system("rm -rf /home/$user");
56 } else {
57     print "Użytkownik wykorzystuje niestandardowy folder macierzysty.\n";
58     print "Usuń go ręcznie.\n";
59     print "Folder = $info[7]\n";
```

Kod dokonuje bardzo ważnego sprawdzenia. Jeśli użytkownik korzystał z niestandardowego folderu macierzystego, skrypt go nie usuwa. W ten sposób unika się problemu `sccs`. Problem ten występuje wtedy, gdy administrator odkrywa użytkownika o nazwie `sccs`, który nigdy się nie logował. W takiej sytuacji próbuje usunąć jego konto.

Pierwszym krokiem jest bardzo często usunięcie folderu domowego danego użytkownika z zastosowaniem polecenia:

```
# rm -rf ~sccs
```
(*Nie wykonuj tego polecenia!*)

Okazało się, że `sccs` było kontem systemowym, a folder macierzysty był ustawiony na /. Oznaczało to, że usunięcie folderu macierzystego było równoznaczne z wykonaniem operacji:

```
# rm -rf /
```

Jeśli kogoś to polecenie nie przeraża, zapewne nie zna dobrze systemu Unix. Polecenie usuwa zawartość całego dysku twardego. Na szczęście administrator miał kopie zapasowe i wyrozumiałą żonę, która nie była zła, że przyszedł do domu dopiero o 3 w nocy (odtwarzanie danych zajmuje sporo czasu!).

Aby uniknąć problemu sccs, skrypt usuwa jedynie te foldery, które znajdują się w bezpiecznym miejscu. Gdy sytuacja jest niepewna, skrypt pomija usuwanie folderu macierzystego, pozostawiając to zadanie administratorowi.

Ostatnia uwaga: ostatnim edytowanym plikiem jest */etc/passwd*. Wynika to z faktu, iż jest to plik z założoną blokadą w momencie usuwania lub dodawania użytkowników. Gdy nazwa pliku zostaje zmieniona w trakcie procesu edycji, blokada zostaje zdjęta. Z tego względu plik musi być edytowany na samym końcu.

Modyfikacje skryptu

Oczywiście istnieją inne programy, które potrafią lepiej usuwać użytkowników z systemu niż przedstawiony skrypt. Jeśli jednak trzeba usunąć dużą liczbę użytkowników, przedstawione podejście pozwoli zaoszczędzić mnóstwo czasu.

Skrypt 27. Zabicie zablokowanego procesu

Pracowałem dla dużej firmy, która wykorzystywała najgorszy system kompilacji i generowania, jaki kiedykolwiek widziałem. Jeden z największych problemów polegał na tym, że gdy wylogowywało się bez poprawnego zamknięcia środowiska projektowania aplikacji, jeden z programów tła mógł zostać zablokowany przez ciągłe próby połączenia się z niedziałającą aplikacją główną.

W ten sposób bardzo wydajne komputery zajmujące się kompilacją programów były spowalniane przez zbędne procesy. Trzeba było szukać użytkownika lub administratora systemu, by unicestwił zablokowany proces.

Perl zapewnia automatyczne wykonywanie zadań, które tradycyjnie trzeba wykonywać ręcznie — w tym przypadku dotyczy to identyfikacji i wyłączania zablokowanych programów.

Kod

```
1 #!/usr/bin/perl
2 use strict;
3 use warnings;
4 #
5 #  Wyłącza zablokowane procesy.
6 #
7 #  Zablokowany proces to taki, który zajął procesowi
8 #  ponad godzinę.
9 #
10 #  UWAGA: Program został zaprojektowany do przyjaznego wyłączania aplikacji.
11 #  Najpierw wysyła przyjazny sygnał zakończenia (SIGTERM) do zablokowanego
12 #  procesu, prosząc go o wyłączenie się. Jeśli zmieni się wysyłany sygnał
13 #  na "zabójstwo z premedytacją" (SIGKILL), proces zostanie ZMUSZONY
14 #  do zakończenia działania.
```

```
#
# Zakończenie aplikacji musi zostać potwierdzone przez operatora.
#
# Jeśli okaże się, że pewien użytkownik często pozostawia zablokowane
# procesy, można zapewnić usuwanie jego programów bez
# konieczności potwierdzania tej operacji.
#
my $max_time = 60*60;     # Maksymalny czas rzeczywistego działania
                          # podawany w sekundach.

# Nazwy procesów, które mogą działać dłuższy czas.
my %exclude_cmds = (
    # Pomijamy elementy KDE, które potrafią zajmować mnóstwo czasu.
    'kdeinit:' => 1,
    '/usr/bin/krozat.kss' => 1
);
# Użytkownicy, których procesów nie należy usuwać.
my %exclude_users = (
    root => 1,
    postfix => 1
);
# Wykorzystaj polecenie PS do znalezienia zablokowanych programów.
# UWAGA: polecenie ps w wersji dla systemu Linux.
my @ps = `ps -A -eo cputime,pcpu,pid,user,cmd`;
shift @ps;                # Pomiń wiersz nagłówka.
chomp(@ps);

# Przejdź przez każdy proces.
foreach my $cur_proc (@ps) {

    # Pola procesu jako nazwy.
    my ($cputime,$pcpu,$pid,$user,$cmd) =
        split /\s+/, $cur_proc;

    $cputime =~ /(\d+):(\d+):(\d+)/;
    # Czas procesora w sekundach zamiast w wersji sformatowanej.
    my $cpu_seconds = $1*60*60 + $2*60 + $3;

    if ($cpu_seconds < $max_time) {
        next;
    }

    if (defined($exclude_users{$user})) {
        print "Pomijam z racji użytkownika: $cur_proc\n";
        next;
    }

```

```
62      if (defined($exclude_cmds{$cmd})) {
63          print "Pomijam z racji typu procesu: $cur_proc\n";
64          next;
65      }
66
67      # Proces zablokowany. Poproś o jego zakończenie.
68      print "ZABLOKOWANY: $cur_proc\n";
69      print "Zabić? ";
70      my $answer = <STDIN>;
71
72      if ($answer =~ /^[Tt]/) {
73          # Wyłącz łagodnie.
74          kill 'TERM', $pid;
75          print "Wysłany sygnał TERM do procesu\n";
76      }
77  }
```

Uruchomienie skryptu

Skrypt powinien być uruchamiany przez administratora w celu usunięcia zablokowanych procesów.

Wyniki

```
ZABLOKOWANY: mpg123
Zabić? t
Wysłany sygnał TERM do procesu
```

Jak to działa?

Program rozpoczyna się od wykonania polecenia `ps` w celu uzyskania listy procesów.

```
36 # Wykorzystaj polecenie PS do znalezienia zablokowanych programów.
37 # UWAGA: polecenie ps w wersji dla systemu Linux.
38 my @ps = `ps -A -eo cputime,pcpu,pid,user,cmd`;
39 shift @ps;   # Pomiń wiersz nagłówka.
40 chomp(@ps);
```

Następnie przechodzi przez każdy proces w celu jego sprawdzenia.

```
42 # Przejdź przez każdy proces.
43 foreach my $cur_proc (@ps) {
```

Rozbija poszczególne pola, aby mieć do nich lepszy dostęp.

```
45      # Pola procesu jako nazwy.
46      my ($cputime,$pcpu,$pid,$user,$cmd) =
47          split /\s+/, $cur_proc;
```

Czas procesora jest podawany w formacie HH:MM:SS. Należy go zamienić na format bardziej wygodny w obliczeniach

```
49      $cputime =~ /(\d+):(\d+):(\d+)/;
50      # Czas procesora w sekundach zamiast w wersji sformatowanej.
51      my $cpu_seconds = $1*60*60 + $2*60 + $3;
```

Po wyliczeniu czasu skrypt sprawdza, czy analizowany program przekroczył dopuszczalny czas działania.

```
53      if ($cpu_seconds < $max_time) {
54          next;
55      }
```

Są użytkownicy, których procesy są pomijane w trakcie analizy.

```
57      if (defined($exclude_users{$user})) {
58          print "Pomijam z racji użytkownika: $cur_proc\n";
59          next;
60      }
```

Podobnie jest w przypadku wybranych programów. Je także należy pominąć w trakcie analizy.

```
62      if (defined($exclude_cmds{$cmd})) {
63          print "Pomijam z racji typu procesu: $cur_proc\n";
64          next;
65      }
```

Jeśli proces przeszedł poprawnie wszystkie testy, można go interaktywnie wyłączyć.

```
67      # Proces zablokowany. Poproś o jego zakończenie.
68      print "ZABLOKOWANY: $cur_proc\n";
69      print "Zabić? ";
70      my $answer = <STDIN>;
```

```
71
72      if ($answer =~ /^[Tt]/) {
73          # Wyłącz łagodnie.
74          kill 'TERM', $pid;
75          print "Wysłany sygnał TERM do procesu\n";
76      }
77  }
```

Modyfikacje skryptu

Skrypt silnie zależy od danych uzyskiwanych z polecenia `ps`. Wynik działania tego polecenia zależy od stosowanego systemu operacyjnego. Należy dostosować skrypt do własnego systemu.

Zabijanie procesów nie jest tylko procedurą techniczną, ale zależy również od polityki firmy w kwestii zablokowanych procesów innych osób. Gdy polityka firmy dopuszcza ich usuwanie, warto skorzystać z przedstawionego skryptu.