Systemy operacyjne

Laboratorium 8

Perl – find

Jarosław Rudy
Politechnika Wrocławska

28 lutego 2017

Temat obejmuje przeszukiwanie drzew katalogowych z użyciem `perl`a oraz podstawowe zdolności w używaniu referencji, tablic asocjacyjnych i mechanizmów typu `stat`.

1 Wstęp

`Perl`owe "przeszukiwanie" drzew katalogów na podobieństwo komendy `find` odybwa się z wykorzystaniem modułu `File::Find`, który należy dołączyć w początkowej części skryptu:

```
use File::Find;
```

Samo przeszukiwanie zaczyna się od wywołania jednej z dwóch funkcji: `find` lub `finddepth`. Wbrew nazwie obie funkcją przeszukują drzewo metodą "wgłąb". Tzn. dla poniższego drzewa:

```
      A
    /   \
   B     E
  / \   / \
 C   D F   G
```

całe poddrzewo `B` zostanie przetworzone przed poddrzewem `E` albo odwrotnie. Różnica pomiędzy `find` i `finddepth` polega na tym, że pierwsze przetwarza katalog *przed* jego zawartością, zaś `finddepth` *po* jego zawartości. Pierwszy da więc kolejność np.:

```
A B C D E F G
```

zaś drugie może dać kolejność:

```
C D B F G E A
```

żadne jednak nie da kolejności:

```
A B E C D F G
```

ani

```
C D F G B E A
```

które były by typowe dla poszukiwania "wszerz".

Samo użycie funkcji `find` (lub `finddepth`) ma następującą bazową składnię:

```
find( \&funkcja_przetwarzania , @katalogi );
```

Drugi argument, `@katalogi`, określa które katalogi należy przetworzuć. Może to być pojedynczy katalog (skalar lub tablica jednoelementowa) lub wiele katalogów (tablica wieloelementowa). Argument pierwszy określa z kolei nazwę funkcji, jaką należy wywołać dla każdego przetwarzanego elementu (z pewnymi wyjątkami). Sam zapis `\&` oznacza kontekst "referencji kodu" i jest niejako odpowiednikiem wskaźnika na funkcję z języka `C/C++`. Aby kod mógł zadziałać, należy w programie zdefiniować samą funkcję w postaci:

```
sub funkcja_przetwarzania()
{
   (ciało funkcji)
}
```

Oczywiście, czy użyjemy nazwy `funkcja_przetwarzania`, `wanted` czy dowolnej innej nie jest istotne.

Powyższe wywołanie `find` wystarczy do podstawowych zastosowań, ale istnieje dodatkowa forma wywołania:

```
find( %options , @directories );
```

gdzie `%options` jest pewnym *haszem*, który definiuje parametry potrzebne funkcji `find`.

2 Hasze

Hasze w `perlu` są realizacją tablic asocjacyjnych (map, słowników) przechowujących dane na zasadzie klucz-wartość. W `perlu` mamy więc "natywnie" dostęp do zwykłych tablic (kontekst `@`) oraz do tablic asocjacyjnych (kontekst `%`). Poniżej pokażemy podstawową składnię związaną z tworzeniem i wykorzystaniem haszy. Definicja hasza wygląda następująco:

```
%hash = ( 'klucz1' , 'wartosc1' , 'klucz2' , 'wartosc2' );
```

lub

```
%hash = ( 'klucz1' => 'wartosc1' , 'klucz2' => 'wartosc2' );
```

Oczywiście możemy użyć cudzysłowów zamiast apostrofów lub podać wartości z użyciem zmiennych:

```
%hash = ( $zmienna => "wartosc" );
```

Warto zwrócić uwagę na fakt, że nawiasy w tej formie są nawiasami *okrągłymi*. Możemy również dodać (lub nadpisać) wartość dla danego klucza w istniejącym haszu:

```
$hash{ 'klucz' } = 'wartosc';
```

Zauważmy, że w tym przypadku a) odwołujemy się do pojedynczego klucza, więc używamy *dolara* zamiast procenta oraz b) stosujemy nawiasy *klamrowe*.

Funkcja (operator) `keys` pozwala w łatwy sposób uzyskać dostęp do wszystkich kluczy i wartości hasza:

```
for ( keys %hash )
{
   print "klucz to " . $_;
   print "wartosc to " . $hash{ $_ };
}
```

Wrócmy teraz do funkcji `find`. Hasz `%options` powinien definiować wartości dla pewnego zestawu kluczy. Jeden klucz jest obowiązkowy – jest nim klucz `'wanted'`, dla którego wartość musi zawierać referencję kodu (funkcji) do wywołania. Wartości pozostałych kluczy pozwalają sterować takimi zachowaniami funkcji `find` jak podążanie za linkami symbolicznymi, automatyczną zmianą `CWD` po napotkaniu podkatalogu itp. Pełny opis można znaleźć na stronie perldoca.

3 I-węzły, `stat` i "magiczny" uchwyt

Wewnątrz naszej funkcji zdefiniowanej poprzez referencję typu `\&wanted` należy umieścić kod dokonujący faktycznego przetwarzania. Innymi słowy, testy z rodzaju `-type` czy `-name` lub akcje typu `-print` i `-exec` nie są w `perl`u zdefiniowane i należy po prostu napisać konkretny algorytm przetwarzania z wykorzystaniem narzędzi `perl`owych. Jednakże przetwarzanie plików zwykle wymaga dostępu do informacji z i-węzła pliku. Informacje te są bazowo dostarczane po użyciu na pliku `perl`owej funkcji `stat`. Uwaga! Funkcja ta powinna być dostępna od razu – dołącznie do programu modułu `File::stat` zamienia domyślną funkcję `stat` na jej inną wersję.

Po użyciu `stat` otrzymamy 13-to elementową listę zawierającą informację o pliku. Dostęp do poszczególnych "pól" listy można przeprowadzić na kilka sposobów (informacje do znalezienia w Internecie), w instrukcji skupimy się jednak na kilku szczegółach i pułapkach dotyczących użycia tej listy.

Po pierwsze, jeden z elementów listy zawiera jednocześnie dwie informacje: typ pliku oraz prawa dostępu. Jest to zachowanie podobne do komend typu `ls`, które raportują prawa wraz z typem pliku np.:

```
drwxrwxrwx
```

Dodatkowo, `perl`owa funkcja `stat` przechowuje te dane jako *jedną* liczbę w systemie *dziesiętnym*! Poprawne zbadanie praw dostępu wymaga więc trochę uwagi i wykorzystania np. operatorów bitowych.

Drugą kwestią jest różnica pomiędzy `stat` oraz `lstat`. Jeśli testujemy dowiązanie symboliczne, to pierwszy zwróci informacje odnośnie i-węzła właściwego dla *celu* dowiązania (a właściwie końca "łańcucha" dowiązań), a drugi – właściwe dla *samego dowiązania*.

Możemy używać również wielu operatorów plikowych na podobieństwo tych znanych z `bash`a:

```
if ( -f $sciezka )
```

Operatory te opisane są na stronie perldoc po wyszukaniu hasła `-x`. Nietypowe są testy `-M`, `-A`, `-C`, które liczą czas od rozpoczęcia skryptu w DNIACH (zmiennoprzecinkowo), a nie w liczbie sekund od tzw. epoki (1 stycznia 1970).

Ostatnia uwaga dotyczy wydajności. Wywołanie `stat` zapisuje wynik (informacje o i-węźle) w odpowiednim miejscu w pamięci. Co więcej identyczna sytuacja ma miejscu w sytuacji, gdy wykorzystujemy operatory plikowe typu `-f` czy `-d` – nadpiszą one to samo miejsce w pamięci. Jeśli każde wywołanie "statuje" osobny plik, to sytuacji nie można poprawić. Jeśli jednak statujemy ten sam plik, to chcemy uniknąć zapisów typu:

```
stat $sciezka;
if ( -f $sciezka || -d $sciezka )
```

które powodują 3-krotne zapisanie do pamięci informacji o tym samym i-węźle. Rozwiązaniem może być użycie "magicznego uchwytu" danego symbolem `"_"`. Użycie go zamiast ścieżki (zarówno dla `stat`, `-f`, `-d` i innych operatorów plikowych) spowoduje odwołanie się do *istniejącej* już zapisanej struktury i-węzła w pamięci. Przykład poprzedni możemy wtedy zapisać jako:

```
stat $sciezka;
if ( -f _ || -d _ )
```

Ten trik należy jednak stosować ostrożnie, pamiętając o kilku szczegółach:

1. "Magiczny" uchwyt _, a "magiczna" zmienna `$_` to NIE jest to samo!
2. Domyślnym argumentem dla `stat` i operatorów typu `-f` jest `$_`, a nie _! Tak więc jeśli chcemy użyć uchwytu _, to trzeba go podać jawnie (z wyjątkiem cukierka składniowego opisanego poniżej)!
3. Użycie "magicznego" uchywtu jest uzasadnione tylko za "drugim i kolejnym podejściem" tzn. gdy mamy pewność, że w pamięci znajdują się już informacje o pliku, który nas interesuje. Zwykle oznacza to, że pierwsze użycie `stat` lub operatorów typu `-f` musi odbyć się z podaniem ścieżki lub "normalnego" uchwytu. Jest to szczególnie istotne, jeśli `stat` lub operatory plikowe wywołane są pod warunkiem, należy wtedy zdawać sobie sprawę czy dany plik był "statowany" czy nie.

4. Pamiętajmy, że `lstat` i `-l` "statują" link symboliczny, a nie jego cel! Oznacza to, że użycie `-l _` po `stat $sciezka` jest zwykle błędem, bo potencjalnie prowadzi to do "statowania" dwóch *różnych* plików.

Dodatkowo, w nowszych wersjach `perl`a istnieje cukierek składniowy. Poniższy kod:

```
-f $sciezka && -r _ && -x _
```

może zostać zapisany jako:

```
-f -r -x $sciezka
```

Należy zauważyć, że działa to tylko dla operatora `AND` (`&&`), a nie dla `OR` (`||`). Oczywiście, jeśli `$sciezka` została już wcześniej "zestatowana", to możemy od razu użyć "magicznego" uchwytu:

```
-f -r -x _
```