Könyvtárkezelés Perlben

Általánosan elfogadott nézet, hogy amit C-ben nem lehet megoldani, azt más nyelvben sem. Mára újabb egyetemes tétellel gazdagodtunk: amit Perlben nem lehet egyszerűen megoldani, azt nem lehet más nyelvben sem.

A fenti állítást bizonyítandó nézzük meg közelebbről a könyvtárkezelés Perl-beli megvalósítását. Mint látni fogod, semmivel sem bonyolultabb könyvtárlistát készíteni, mint állományt kiíratni. Szemügyre vesszük a könyvtármutatókat, szó lesz egy objektumközpontú felületről, továbbá a jogosultságok és a tulajdonos beállításáról.

Az állománytípusok

Kezdjük az elején! Egy könyvtárlista elemei nem egységesen fájlok, hanem típusuk szerint különbözhetnek. A közvetett hivatkozástól (symbolic link) a könyvtáron át egészen a hálózati foglalatig számos fajtát határozhatunk meg. A Perlben minden típusnak megfelelően létezik egy műveleti jel (operator), ami igazat ad vissza, ha az elem az adott típushoz tartozik, ellenkező esetben hamisat. Hasonló ez a bashből ismert módszerhez, a két nyelv műveleti jelei között vannak azonban eltérések. Az alábbi egyszerű függvény egy skalárt ad vissza, ami a megadott állomány típusát jellemzi.

```
sub fileType {
     my $file = shift;
     return "szimbolikus link" if -l $file;
     return "konyvtar" if -d $file;
     return "fifo" if -p $file;
     return "karakter-specialis" if -c $file;
     return "blokk-specialis" if -b $file;
     return "socket" if -S $file;
     return "szoveg" if -T $file;
     return "binaris" if -B $file;
     return "ismeretlen";
}
```

Így egyszerűen meghatározhatod egy állomány típusát, a legkönnyebben az alábbi szemléletes sorral foghatod munkára az eljárást:

```
print fileType "/dev/hda";
```

Természetesen a nyelv rugalmasságából eredően állománynév helyett fájlmutatót is megadhattunk volna, most azonban az egyszerűség kedvéért a könnyebb utat választottam.

A könyvtármutatók

Mint azt az elején említettem, könyvtárlistát gyerekjáték kiíratni. Nem elég, hogy nem nehezebb egy állomány tartalmának a megjelenítésénél, de egy az egyben ugyanazt az eljárást igényli. A szemmel látható különbség mindössze a függvények neveiben jelentkezik. A végletekig leegyszerűsítve azt mondanám, hogy csupán minden, az állománykezelésnél megismert függvény neve mögé egy `dir`-t kell ragasztani. Az `opendir` függvény segítségével megnyitjuk a könyvtárat, a későbbiekben az értékként átadott könyvtármutatóval fogunk hivatkozni rá. A gyémánt műveleti jel (< >) megfelelője ebben az esetben a `readdir`. Végül a `closedir` segítségével töröljük a könyvtármutatót a névtérből. Lássuk mindezt a gyakorlatban!

```
my ($dir,$file,$type);
($dir = $ARGV[0]) or die "Hasznalat: ".$0."
                         ➥<konyvtar>\n";
opendir (DIR, $dir) or die $dir.
       ➥": ".$!."\n";
print $dir." tartalma:\n";
while ($file = readdir(DIR)) {
     $type = fileType($dir."/".$file);
     print "\t".$file." (".$type.")\n";
}
closedir (DIR);
```

A fenti program a korábban már tárgyalt `fileType` függvényt használja az állomány típusának a meghatározásához, és egy egészen tetszetős kimenetet hoz létre. Mint láthatod, nincs különbség az állománykezeléshez képest, illetve csak igen csekély. Egy könyvtármutatót nem lehet írási műveletre felhasználni. Továbbá a `readdir` nem használja az alapértelmezett változót ($_), így egyet kötelező megadni.
Érdemes megjegyezni, hogy a `readdir` a gyémánt műveleti jelhez hasonlóan tömb és skalár összefüggésben is használható. Ez azt jelenti, hogy a kiíratás úgy is megoldható, hogy egyetlen `readdir` hívással egy `@files` tömbbe az összes bejegyzést begyűjtöd, majd egy `for $file (@files) { }` szerkezettel végigléptetted az elemeken. Az előbbi ciklusban minden egyes lefutásnál a `$file` a következő értéket veszi fel.
Ne felejtsd el, hogy a könyvtárlista tartalmazni fogja a `'.'` és a `'..'` (a pont és a két pont) elemeket is. Amennyiben ezekre nem vagy kíváncsi, a ciklusban egy mintaillesztéssel szűrd ki őket:

```
next if $file =~ /^\.\.?$/;
```

A DirHandle modul

Az alapkönyvtár része az említett modul, ami objektumközpontú felületként szolgál a már megismert alacsonyabb szintű függvényekhez. Ennek értelmében a `DirHandle` használata nem jelent több szolgáltatást, mindössze más megközelítése ugyanannak a feladatnak. Lássunk egy példát!

```
use DirHandle;

my ($dir,$dh,$file,$type);
($dir = $ARGV[0]) or die "Hasznalat: ".$0."
                         ➥<konyvtar>\n";
($dh = new DirHandle ($dir)) or die $dir.
       ➥": ".$!."\n";
```

```
print $dir." tartalma:\n";
while ($file = $dh->read) {
    next if $file =~ /^\.\.?$/;
    $type = fileType($dir."/".$file);
    print "\t".$file." (".$type.")\n";
}
$dh->close;
```

Pillanatnyi könyvtár

Egy parancsértelmezőhöz hasonlóan a Perl is ismeri a pillanatnyi könyvtár fogalmát. Ez alapértelmezésben az a könyvtár, ahonnan a parancsfájlt indítottad. Nem feltétlenül egyezik meg azzal a hellyel, ahol a program található. Erre különös tekintettel figyelj! A legjobb, ha ahol csak lehet, abszolút elérési utat használsz.
A jelenlegi munkakönyvtárad lekérdezhető a `Cwd` modul részét képező `cwd` függvénnyel. A `chdir` pedig – ahogy a nevéből is sejtheted – a pillanatnyi könyvtár megváltoztatására használható. Ezek egy bashből kiadott `pwd`, illetve `cd` parancshoz hasonlóan viselkednek.

A umask függvény

Egy állomány létrehozása esetén a jogosultságokat tekintve a Perl eredeténél fogva Unix-szerűen viselkedik. Létezik egy alapértelmezett maszk, amit az új állomány jogainak meghatározásához használ fel. Ezt hívják `umask`-nak. Röviden szólva ez egy olyan nyolcas számrendszerbeli szám, ami megmutatja a megvont jogokat. Ez az érték jellemzően 022. Ez azt jelenti, hogy minden engedélyezett, leszámítva a „csoport” és a „többiek” írási jogát. Így az újonnan létrehozott állományok jogosultsága 644, a könyvtáraké 755.
A Perl `umask` függvénye nyújt lehetőséget az említett alapértelmezett maszk megváltoztatására. Azt például, hogy az újonnan létrehozott állományokra minden engedélyezve legyen, az alábbi függvényhívással érheted el:

```
umask 0000;
```

A chmod függvény

A `chmod` segítségével explicit módon is megadhatod egy állomány vagy könyvtár elérési jogait. A függvény értékként egy listát vár, aminek az első eleme az új jogosultság nyolcas számrendszerben megadva, a további elemek pedig állományok. Ha gondban vagy a nyolcas számrendszerrel, az `Fcntl` modullal számok helyett beszédesebb állandókat használhatsz. Íme a példa:

```
use Fcntl ':mode';
chmod S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH|S_IXOTH,
"alkalmazas.pl";
```

A fenti állandók értelmezése:

```
S_IRWXU = S_I Read Write eXecute User
➥(a tulajdonosnak olvasás, írás, futtatás)
S_IRGRP = S_I Read GRouP
➥(a csoportnak olvasás)
S_IXGRP = S_I eXecute GRouP
➥(a csoportnak futtatás)
S_IROTH = S_I Read OTHer
➥(a többieknek olvasás)
S_IXOTH = S_I eXecute OTHer
➥(a többieknek futtatás)
```

Ezeket bitenként egy VAGY művelettel összegeztük, így ugyanazt kaptuk, mintha 0755-öt adtunk volna meg.

A chown függvény

A `chown` segítségével megváltoztathatod egy, esetleg több könyvtár vagy állomány tulajdonosát, illetve csoportját.
A függvény a `chmod`-hoz hasonlóan egy listát vár. Ennek első eleme az új felhasználói azonosító, a második a csoportazonosító, a további értékek állománynevek. Természetesen lehetőség nyílik csak az egyik, illetve a másik megváltoztatására, ekkor annak az azonosítónak a helyén, amit nem kívánunk módosítani, `-1`-et kell megadnunk.
Kényelmetlen azonban számokkal dolgozni – a Perl itt két újabb függvénnyel áll a rendelkezésünkre. A `getpwnam`, illetve a `getgrent` a */etc/passwd*-ből és a */etc/group*-ból nyerik ki a névhez tartozó azonosítót. Az alábbi példa a ***szoveg.txt*** tulajdonosát `balazs`-ra változtatja:

```
chown getpwnam("balazs"), "-1", "szoveg.txt";
```

Az mkdir, az rmdir és az unlink

A bámulatosan beszédes nevű függvények pont azt teszik, amit a parancsértelmező által futtatott programok is tennének.
Az `mkdir` új könyvtárat hoz létre; kötelező érték a létrehozandó könyvtár neve, ezután a jogosultság is megadható. Az utóbbi elhagyása esetén a `umask`-nak megfelelően jön létre a könyvtár.

```
mkdir "appz", 0755;
```

Az `rmdir` egy könyvtárat töröl. Értékként a könyvtár neve szerepel. Fontos, hogy a rendszerből ismert testvéréhez hasonlóan csak akkor szüntet meg egy könyvtárat, ha az üres.

```
rmdir "appz";
```

Az `unlink` állományokat töröl. Értékként a törlendő állományok neveivel egy listát vár.

```
unlink "alma.txt", "korte.txt", "szilva.txt";
```

Miért foglalkoztunk ezekkel?

Miért érdemes ezzel foglalkozni, ha ott van a Midnight Commander? Azért, mert mindennapos feladat lehet egy könyvtár alatti összes könyvtárnak adott jogosultságúra, míg az összes állománynak más jogosultságúra történő beállítása. Ezt kézzel tíz bejegyzés esetén még az őrület elkerülésével meg lehet oldani, ezer esetén már aligha. Sokszor eszedbe juthat, hogy erre programot írni felesleges, gyorsabban végzel, ha kézzel csinálod. Szerintem ez súlyos tévedés. Egyrészt legközelebb is ütközhetsz ilyen feladatba, másrészt a tapasztalat azt mutatja, hogy hamarabb végzel egy parancsfájllal, mint a nyílbillentyűkkel az mc-ben. Arról nem is beszélve, hogy a számítógépet pontosan erre találták ki: a feladatok önműködővé tételére.
Kellemes programozást!

Fülöp Balázs (xut@freemail.hu)
18 éves, imádja a Túró Rudit, a Debian Linuxot és a teheneket. Az ELTE Radnóti Miklós Gyakorlóiskola tanulója immár ötödik éve. Kedvenc írója Slawomir Mro□ek. Leginkább a számítógépes hálózatok biztonsága érdekli.