© Kiskapu Kft. Minden jog fenntartva

Héjprogramozás Linux alatt (7. rész)

Akármilyen nyelven fejlesztünk is programot, egy idő után számos olyan építőelemet fedezünk fel a saját munkánkban, amit újra és újra felhasználunk.

Mostanra e sorozatunkban is eljutottunk odáig, hogy elkezdhetjük áttekinteni a héjprogramozás kifinomultabb módszereit, valamint a programban gyakran felbukkanó, „újrahasznosítható” kódelemeket.

A parancssori kapcsolók kezelése

A sorozat egy korábbi részében már megtanultuk, hogyan lehet a parancssori kapcsolókat kezelni. Tudjuk tehát, hogy ezekhez a `$1`, `$2` stb. szimbólumok segítségével férhetünk hozzá. Mármost a parancssori kapcsolók is alapvetően két csoportra oszthatók: vannak az „igazi” kapcsolók, amelyek a program által megvalósított feldolgozási művelet célját vagy forrását határozzák meg, és léteznek az értékek, amelyek a feldolgozás miként jét szabályozzák. Tekintve, hogy Unix alatt minden fájlnak minősül, az értékek általában fájlnevek, a kapcsolók pedig egybetűs parancsok, amelyeket a Unix hagyományainak megfelelően általában egy – (mínuszjel) előz meg.

Ahhoz, hogy programunk a kapcsolók hatására a megfelelő műveletsort hajthassa végre, rendelkeznie kell egy olyan résszel, amely a kapcsolókat felismeri és a végrehajtást a megfelelő mederbe tereli.

Tegyük fel, hogy programunk működése legfeljebb három különböző kapcsolóval vezérelhető. Ezek a `-a`, `-b` és `-c` szimbólumok legyenek. Ezenkívül a kapcsolók, illetve a fájlnevek megadásának sorrendjére vonatkozóan semmiféle további megkötéssel nem élünk: például a kapcsolókat egybe szabad írni, vagyis a `-a -b` formátum helyett azonos jelentéssel használható a `-ab` is. Továbbá egy vagy több fájlnevet is meg lehet adni kapcsoló előtt és után, vagy akár két kapcsoló között is, sőt egy kapcsoló akár többször is szerepeltethető. Összefoglalva tehát, a felhasználó gyakorlatilag azt tesz, amit akar, a programnak kell elég intelligensnek lennie ahhoz, hogy bármilyen formátumú parancssort helyesen értelmezzen. (Egyetlen dolgot azért mégis kössünk ki: fájl neve soha ne kezdődjön mínuszjellel.)

Milyen elemekből kell tehát állnia egy ilyen programnak? Először is a megoldandó feladat két, világosan elkülönülő részre választható szét: azonosítanunk kell a kapcsolókat, és össze kell gyűjtenünk a fájlneveket. Ha két művelet ennyire élesen elkülönül, célszerű két függvény formájában megvalósítani őket.

A parancssorba a felhasználó által „beömlesztett” értékek szétválogatásának alapelve nyilván az lehet, hogy a kapcsolók vagy azok csoportjai mínuszjellel kezdődnek, a fájlnevek pedig nem. Ha fájlnevet találunk valahol, azt elegendő egy nagy listába begyűjteni, ami viszonylag könnyen megoldható, ha tudjuk, hogy héjprogramokban a karakterláncok összemásolásához nem kell egyebet tennünk, mint egy értékadás jobb oldalán egymás mellé írni őket.

A kapcsolóknál eleve fel kell készülünk rá, hogy a felhasználó esetleg csoportokban adta meg őket, így a legegyszerűbb megoldás az, ha egy-egy `grep` parancs segítségével minden mínuszjellel kezdődő parancssori kapcsolóban az összes lehetséges kapcsolót végigvizsgáljuk, illetve az esetlegesen jelenlevő érvénytelen kapcsolókat is. Ennyi bevezető után lássuk a feladat egyik lehetséges megoldását!

```
 1: #!/bin/sh
 2: PROGRAMNEV=`basename $0`
 3:
 4: # Kapcsolók kigyűjtése
 5: kapcsolok()
 6: {
 7:   for i in $*
 8:   do
 9:     if echo $i | grep ^- > /dev/null
10:     then
11:     # Hibás kapcsoló keresése
12:       if echo $i | grep '[^-abc]'
                      ➥> /dev/null
13:       then
14:         return 1
15:       fi
16:     # Érvényes kapcsolók keresése
17:       for kapcsolo in a b c
18:       do
19:         if [ `echo $i | grep $kapcsolo |
                  ➥wc -l` -eq 1 ]
20:         then
21:           parancs="$kapcsolo=1"
22:           eval "$parancs"
23:         fi
24:       done
25:     fi
26:   done
27: }
28:
29: # Fájlnevek kigyűjtése
30: fajlnevek()
31: {
32:   lista=""
33:   for i in $*
34:   do
35:     if echo $i | grep -v ^- > /dev/null
36:     then
37:       lista=$lista" "$i
38:     fi
39:   done
40:   echo $lista
41: }
42:
43: # Tájékoztató szöveg
44: if [ $# -eq 0 ]
45: then
46:   echo "Használat: $PROGRAMNEV [-abc]
            ➥<fájlnév>"
47:   exit 1
48: fi
49:
50: # Kapcsolók vizsgálata
```

```
51: a=0; b=0; c=0
52: kapcsolok $*
53: if [ $? -ne 0 ]
54: then
55:   echo "Hibás kapcsoló!" ;  exit 1
56: fi
57:
58: # Fájlnevek kigyűjtése
59: fajlok=`fajlnevek $*`
60:
61: echo "Kapcsolók beállításai: a=$a b=$b
        ➥c=$c"
62: echo "A feldolgozandó fájlok listája:
        ➥$fajlok"
```

A kapcsolókat a `kapcsolok()`, a fájlok neveit pedig a `fajlnevek()` függvény kezeli. Bár a főprogramban ezt hívjuk meg később, kezdjük a vizsgálatot az utóbbival. Mindenekelőtt figyeljük meg, hogy a függvény meghívásakor (59. sor) a program teljes paraméterkészletét (`$*`) átadjuk neki. Erre azért van szükség, mert – mint arról korábban már volt szó – a függvény „program a programban”, vagyis mindenből sajátja van. Létezik saját bemenete és kimenete, valamint saját parancssora is. Ha ide nem emeljük át az egész héjprogramnak átadott értékeket, akkor a függvény nem fogja látni őket. Számára ugyanis `$1` nem a főprogram első parancssori értékét jelenti, hanem a sajátját.
A végrehajtandó műveletsor lelke a 35. sorban látható. Itt megvizsgáljuk, hogy a 33. sorban induló `for` ciklus által pillanatnyilag kiválasztott érték első karaktere mínuszjel-e vagy sem. Ha nem (a negálást a `grep -v` kapcsolója végzi), akkor a tartalmát egy szóközzel kiegészítve a `lista` nevű gyűjtőváltozóhoz másoljuk. A ciklus lefutása után ennek a tartalmát adjuk vissza egy közönséges `echo`-val a függvény szabványos kimenetén keresztül. Megint fontos hangsúlyozni, hogy a függvény csatornájáról van szó, vagyis a fájlok listája nem a képernyőre kerül. Éppen ennek a kimenetnek az elfogására, és a `fajlok` nevű változóban való elhelyezésére szolgál az 59. sorban a parancsbehelyettesítés.
A kapcsolók vizsgálatánál hasonló módszereket alkalmazunk. Ha egy kapcsoló meg van adva a parancssorban, akkor egy, a nevével azonos héjváltozó értékét 1-re állítjuk. (Természetesen bármilyen más módszer is használható lenne.) A `kapcsolok()` függvénynek átadott értékeken a 7. sorban induló `for` ciklus megy végig. Minden mínuszjellel (-) kezdődő csoportnál először célszerűen azt kell megvizsgálnunk, hogy tartalmaz-e téves kapcsolót (`grep '[^-abc]'`). Figyeljük meg, hogy itt és valamennyi ehhez hasonló vizsgálatnál a `grep` kimenetét a */dev/null*-ba irányítottuk. Erre azért van szükség, mert a `grep` alapértelmezett viselkedése szerint a találatokat kiküldi a szabványos kimenetére. Ez most azonos lenne a függvény szabványos kimenetével, és mivel az sincs átirányítva, a szöveg végül a képernyőre kerülne. Ha hibás kapcsolót találtunk, azt a függvény azonnali visszatérésével és az 1-es visszatérési értékkel (`return 1`) jelezzük. Ez utóbbit az 53. sorban a `$?` változó vizsgálatával detektálja a főprogram. (A `$?` beépített héjváltozó a legutoljára lefutott parancs visszatérési értékét tartalmazza. Jelen esetben a `kapcsolok()` függvény egyetlen parancsnak minősül.)
A 16–23. sorban immár az érvényes kapcsolókat vizsgáljuk. Egy `for` ciklussal egyenként végigpróbáljuk valamennyit, és ha illeszkedés van az adott csoporton belül, akkor a megfelelő nevű változó értékét `1`-re állítjuk. Ezt az utóbbi műveletet egy kicsit trükkösen kell végrehajtanunk, hiszen most a beállítandó változó nevét maga a ciklusváltozó (`$kapcsolo`) tartalmazza. Többek között az ilyen közvetett értékadások megoldására szolgál az `eval` parancs, amely egy karakterláncban megadott parancssort (jelen esetben a `$parancs`-ot) hajtja végre.

A szabványos megoldás: getopts

A parancssori kapcsolók kezelése annyira általános feladat, hogy maga a Unix, illetve a héj is rendelkezik egy külön erre a célra szolgáló `getopts` nevű paranccsal, ennek használata a következő:

```
getopts "lehetséges_kapcsolók" változónév
```

A `getopts` tulajdonképpen pontosan ugyanazt teszi, mint az előző programunk, vagyis a megadott mintában előforduló betűket keresi a mínuszjellel kezdődő parancssori értékekben. Ha érvényes kapcsolót talál, azt elhelyezi a megadott héjváltozóban. Ilyenkor a visszatérési értéke igaz, egyébként hamis. Ezt az utóbbi tulajdonságát használhatjuk ki arra, hogy egy `while` ciklussal az összes kapcsolót végignézessük vele. (A `getopts` mindig csak a soron következő kapcsolót vizsgálja, de megjegyzi, hogy hol tart.)
Ha hibás kapcsolót talál, akkor alapértelmezés szerint hibaüzenetet küld a képernyőre, a munkaváltozóba pedig egy ?-t (kérdőjelet) tesz. A hibaüzenet kiküldését megtilthatjuk, ha a lehetséges kapcsolók felsorolását egy kettősponttal kezdjük. Mindezek ismeretében előző programunkat a következőképpen is megírhatjuk:

```
 1: #!/bin/sh
 2: # A getopts parancs használata
 3:
 4: PROGRAMNEV=`basename $0`
 5:
 6: if [ $# -eq 0 ]
 7: then
 8:   echo "Használat: $PROGRAMNEV [-abc]
          ➥<fájlnév>"
 9:   exit 1
10: fi
11:
12: while getopts ":abc" KAPCSOLO
13: do
14:    case $KAPCSOLO in
15:      "a") echo "-a kapcsoló megadva";;
16:      "b") echo "-b kapcsoló megadva";;
17:      "c") echo "-c kapcsoló megadva";;
18:      "?") echo "Hibás kapcsoló"; exit 2;;
19:    esac
20: done
```

A rövidség ára a kötöttebb működés. A kapcsolókat továbbra is egybe szabad írni, de a `getopts` azt elvárja tőlünk, hogy a fájlok felsorolását valamennyi kapcsoló megadása megelőzze. Előző, lényegesen hosszabb programunk ugyanakkor e tekintetben is teljesen „liberális” volt.

Büki András (buki.andras@insilico.hu)
Körülbelül kilenc éve dolgozik Linux operációs rendszerrel. Állandó szerzőtársa Prof. Dr. H. V. Kuksinak, akivel a Duna vagy a Tisza partján szoktak az élet és a tudomány viselt dolgairól töprengeni.

© Kiskapu Kft. Minden jog fenntartva