A GNU dbm adatbázis-kezelő

Gyakran megesik, hogy nincs szükség SQL-lekérdezésekre, bonyolult adattáblákra, csak néhány kulcs-érték párra és sebességre. Ilyenkor jöhet jól a GDBM.

Amennyiben C-ben programozol, és égető szükséged lenne néhány változóhoz tartozó érték tárolására, első ötleted bizonyára egy egyszerű szöveges állományban történő tárolás. Ennek azonban több hátránya is van. A szöveges állományban való keresés nagyon lassú, főleg ha nem egy-két rekordról, hanem sokkal többről van szó. Másrészt e módszer használatával lemondhatsz a bináris adatok tárolásáról. Ha igazi programozó vagy, akit nem hagynak nyugodni az ehhez hasonló kérdések, biztosan eltöprengsz rajta, hogyan tudnád valamilyen saját formátumban megoldani a dolgot. Teljesen felesleges, ugyanis a GDBM-et neked találták ki!

A GDBM egy függvénykönyvtár, amellyel a fentebb említett kulcs-érték párok tárolásának gondját lehet megoldani. Ha már programoztál Perlben, biztosan hallottál a hashről. Nos, azok az adatbázisfájlok, amelyeket így hozol létre, szintén hashek. A GDBM, hogy visszafelé is csereszabatos legyen, a régebbi ***dbm***, illetve ***ndbm*** könyvtárak függvényeit is tartalmazza. A GDBM fontos újítása, hogy az elődökkel szemben a segítségével tetszőleges hosszúságú kulcs-, illetve értékpárokat lehet tárolni.

Első lépésként telepítened kell a GDBM-et. Ha Debian-felhasználó vagy, a `libgdbmg1`, illetve a `libgdbmg1-dev` csomagokban található változatot használhatod. Én mégis inkább azt javaslom, hogy a forrást töltsd le, és mindig a legfrissebb változatot fordítsd le. Az általam használt Woodyban még egy 1994-es változat szerepelt. Az igazsághoz az is hozzátartozik, hogy semmi gondom nem volt vele, és ugyanúgy működött, mint a legfrissebb. Mindenesetre a legújabbat ajánlom. A projekt honlapját a ➲ http://www.gnu.org/software/gdbm címen érheted el. Nem túl bőbeszédű, de lényegretörő. Végül is keresned kell egy hozzád közel eső GNU ftp-tükröt, és a ***gdbm*** könyvtárból le kell töltened a legújabb forrást (jelenleg az 1.8.0-s). Magyar ftp-helyet nem találtam, ezért egy osztrák kiszolgálót választottam. Ha lejött, a ***/usr/src*** könyvtárban a következő sorok beírásával csomagold ki:

```
cp gdbm-1.8.0.tar.gz /usr/src
cd /usr/src
zcat gdbm-1.8.0.tar.gz | tar xv
cd gdbm-1.8.0
```

Most jöhet a megszokott `./configure`. A Debian-felhasználók többsége már megszokásból odaírja a végére, hogy `--prefix=/usr`. Ha nem így teszel, a programok többsége a ***/usr/local*** alá települ. A GDBM telepítése folyamán szerintem érthetetlen és zavaró apróság volt, hogy hiába adtam meg a `configure`-nek az előtagot, a `Makefile`-ban továbbra is a ***/usr/local***-ban állt. Akit zavarnak az ilyen finomságok, sajnos kénytelen lesz átírni a `Makefile` 34. sorát.

Ezután a szokásos `make`, majd `make install` következik. Ha programod fordításakor valamit elrontottál, a gcc `undefined reference` hibaüzenettel le fog állni. Ebben az esetben kezdd elölről a folyamatot, és lépésről lépésre ellenőrizd, hogy a leírtaknak megfelelően jártál-e el. Ha elvesznél egy-egy parancs kimeneténeken tengerében, és nem tudod, hogy sikeresen futott-e le, add ki az `echo $?` parancsot. Ez a program legutóbbi visszatérési értékét mutatja. Amennyiben nem nulla, valami hiba történt.

1. lista

```
/*** setmailaddr.c ***/
#include <stdio.h>
#include <string.h>
#include <gdbm.h>
#include <sys/stat.h>

#define ADATBAZIS "emails.db"

int main(int argc,char **argv) {
     datum kulcs,ertek;
     GDBM_FILE dbf;
     printf("%s\n",gdbm_version);
     if (argc!=3) {
       printf("HasznÆlat: %s {nev}
       ➥{email}\n",argv[0]);
       return 1;
     }
     if (NULL==(dbf=gdbm_open(ADATBAZIS,512,
       ➥GDBM_WRCREAT,
       S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH,0))) {
       fprintf(stderr,"Nem siker lt
       ➥megnyitni az adatbÆzist.\n");
       return 1;
     }
     kulcs.dptr=argv[1];

kulcs.dsize=(strlen(argv[1])+1)*sizeof(char);
     ertek.dptr=argv[2];

ertek.dsize=(strlen(argv[2])+1)*sizeof(char);
     if
(gdbm_store(dbf,kulcs,ertek,GDBM_REPLACE)) {
       fprintf(stderr,"Nem siker lt
       ➥eltÆrolni az adatot.\n");
       gdbm_close(dbf);
       return 1;
     }
     printf("%s: %s sikeresen tÆrolva.\n",
             kulcs.dptr,ertek.dptr);
     gdbm_close(dbf);
     return 0;
}
```

A legegyszerűbb, ha egy példán keresztül mutatom be a könyvtár használatát. Írjunk egy egyszerű programot, ami a neveket és a hozzájuk tartozó levélcímeket egy adatbázisban tárolja. Ezután írjuk meg a párját is, ami ugyanezeket az adatokat kiolvassa. Lássuk az elsőt a *listát*, és haladjunk sorról sorra!

Elsőként a szokásos fejlécállományok láthatók. A ***gdbm.h***-ra mindenképpen szükség van, ha a GDBM-et akarod használni. A ***sys/stat.h*** az `open()`-nél már megszokott meghatározások használatához nélkülözhetetlen (lásd később).
A GDBM a kulcsot, illetve az értéket egy-egy szerkezetben tárolja. E szerkezet a ***gdbm.h***-ban van meghatározva:

```
typedef struct {
        char *dptr;
        int dsize;
} datum;
```

A `dptr` mutató az adatra, míg a `dsize` az adat mérete bájtban megadva. Így a `datum`-adattípussal egy kulcs, illetve egy érték-változót hozunk létre.
Az adatbázis megnyitásához egy olyan szerkezet mutatójára van szükség, amit a könyvtár függvényei majd különböző belső célokra használhatnak. Ezt szolgálja a `main` függvény második sora.
Ezután kiíratjuk a használt GDBM-könyvtár változatát. A `gdbm_version` állandó tartalmazza ezt a karakterláncot. Ha nincs meg a két érték (név és e-mail), a program futása megszakad.
Ezt követően az adatbázist a `gdbm_open` függvény segítségével megnyitjuk. Az első érték az állomány neve (ebben az esetben `emails.db`). A második a `block_size`: ez az egyszerre beolvasandó bájtok számát határozza meg, ami legalább 512, illetve ennek valamely egész számú többszöröse. A harmadik érték mondja meg, hogy a folyamat milyen szerepet fog kapni az állománnyal való munka során. A folyamat író vagy olvasó lehet. Ha egy folyamat írja az adatbázist, akkor ugyanabban az időben senki sem fordulhat az állományhoz. Ha egy folyamat olvassa, akkor más folyamatok olvashatják vele párhuzamosan, de senki sem írhatja. Mindezek alapján itt a következő meghatározásokat használhatjuk:

`GDBM_READER`	folyamatolvasó;
`GDBM_WRITER`	folyamatíró;
`GDBM_WRCREAT`	folyamatíró; ha nem létezik, létrehozza;
`GDBM_NEWDB`	folyamatíró; új adatbázist hoz létre akkor is, ha már létezik.

A negyedik érték egy szám, amelynek akkor lesz jelentősége, ha új állományt kell létrehozni, ugyanis ennek jogosultságait jellemzi. A meghatározások jelentéseiért olvasd el az `open` súgóoldalát (`man 2 open`). Én a felhasználónak olvasási-írási jogot adtam, a csoportnak és a többieknek csak olvasásit.
Utolsó értéknek egy függvény adható meg, ami hiba esetén lefut. Ha 0-t adsz meg, a könyvtár alapértelmezett függvénye fut le. A `gdbm_open` visszatérési értéke `GDBM_FILE`-típusú, illetve `NULL`, ha nem sikerült megnyitni.
Ezt követően a `kulcs`, illetve `ertek` szerkezeteket a programnak átadott értékeknek megfelelően töltjük fel. A `dptr` egy az egyben a megfelelő érték mutatója, míg a `dsize` a szöveg hossza. Azért adtam hozzá egyet a `strlen()`-hez, mert a karakterláncot lezáró `\0`-át nem tartalmazza, így az nem kerülne tárolásra.
Most következik a program érdemi része, amely a kulcs-, illetve érték-párt tárolja. A `gdbm_store` értékei: a megnyitott adatbázis, a kulcs, illetve az érték (ezek `datum`-típusúak). Az utolsó érték kétféle lehet:

`GDBM_INSERT`	ha a kulcs már létezik, hibával tér vissza;
`GDBM_REPLACE`	ha a kulcs már létezik, a rekordot felülírja.

Két érték nem szerepelhet egyazon kulccsal az adatbázisban, ezért feltétlenül meg kell határozni, hogy a függvény miként viselkedjen. Ha a függvényt egy olvasó folyamat hívta meg, `-1`-gyel fog visszatérni. Ha a `GDBM_INSERT`-tel hívod meg és a kulcs már létezik, akkor `1`-et ad vissza. Ha minden rendben ment, a visszatérési érték `0`.
Végül értesítjük a felhasználót a sikeres tárolásról, és lezárjuk az adatbázist (`gdbm_close`).
A létrejött állomány egy kulcs esetén is 1,5 Kb, ráadásul „szabad szemmel” olvashatatlan. Ugyanakkor nagyon gyors, és rövidesen meglátod, milyen egyszerűen nyerhető ki belőle az adat. Még egy pillanat erejéig érdemes elidőzni azon, mit mond rá a `file` parancs:

```
emails.db: GNU dbm 1.x or ndbm database,
➥little endian
```

Most jöjjön a második program, amely a tárolt neveket és címeket olvassa ki! (2. lista, 36 CD Magazin/GNUdbm)
Most már csak nagy vonalakban tekintsük át, hogy a program mit is csinál.
A `gdbm_open` negyedik értéke jelenleg `0`, mert az adatbázist olvasásra nyitom meg, újat nem hozok létre.
Ezt követően egy `if-else` szerkezettel eldöntöm, hogy a programnak vannak-e értékei. Ha vannak, a megadott kulcsokhoz tartozó értéket íratom ki, ha nincsenek, az összes kulcsot kiíratom.
Az első ágban egyből új függvényeket láthatsz: `gdbm_firstkey` és `gdbm_nextkey`. A GDBM ezekkel a függvényekkel segíti a programozót, hogy az összes kulcson végig tudjon menni.
A kulcsok egymásutánisága ugyanakkor nem biztosított! Semmi sem határozza meg, hogy milyen sorrendben kell következzenek. Az adatbázis megváltozásával a sorrend másként alakulhat. Ha mindkét programot lefordítottad, adj hozzá új neveket, majd az összeset írasd ki. Látni fogod, hogy amit először vettél fel, nem feltétlenül az első lesz a sorban.
A példa végül is önmagáért beszél, még a `gdbm_fetch`, illetve a `gdbm_exists` függvényekről érdemes egy-két szót ejteni.
A `gdbm_fetch` a megadott kulcshoz tartozó értéket adja vissza a `datum` szerkezetben. Ha a kulcs nem létezik, a visszakapott `datum` szerkezet `dptr` mutatója `NULL` értéket fog kapni. Ezért előtte a `gdbm_exists` függvénnyel érdemes ellenőrizni, hogy a kulcs létezik-e. Fontos még tudni, hogy a `gdbm_fetch` által visszaadott `datum` szerkezet `dptr` mutatója által hivatkozott memóriaterületet a könyvtár a `malloc` függvénnyel foglalja, viszont önműködően nem szabadítja fel. Erre neked, a programozónak kell ügyelned.
Most pedig mindkét programot fordítsd le az alábbi minta szerint:

```
gcc -Wall -o akarmi akarmi.c -lgdbm
```

A `-Wall` az összes figyelmeztetést megjeleníti. Elvileg egyet sem szabadna kapnod. A fordítás után a futtatható állományt a `-o` kapcsoló után álló név alatt érheted el. Végül a `-lgdbm` utasítja a gcc-t, hogy a fordítás végeztével a GDBM könyvtárat fűzze össze. Ez feltétlenül szükséges, különben `undefined reference`-t kapsz!
A sok munka után jöhet a megérdemelt játék. Vegyél fel neveket, írasd ki őket, és figyeld meg, hogyan változik az adatbázis mérete, illetve az elemek sorrendje!
Jó szórakozást!

Fülöp Balázs
(xut@freemail.hu) 17 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.