Clipper-programok Linux alatt (2. rész)

A Clip fejlesztői környezetről szóló sorozatunkat további adatbázis-kezelők ismertetésével folytatjuk.

A clip az Oracle-, PostgreSQL- és MySQL-adatbázisok elérését is lehetővé teszi. Nagy örömmel vettem észre, hogy milyen egyszerű is kedvenc adatbázis-kezelőm, a PostgreSQL elérése a clip programokból. Ennek az illesztésnek a megismerése két szempontból is tanulságos:

- Példát láthatunk rá, hogyan lehet egy összetettebb C-illesztőfelületet a cliphez illeszteni (lásd a ***cliplibs/clip-postgresql*** könyvtár ***pg_clip.c*** forrását).
- Elemezhetjük, hogyan érdemes egy ilyen C nyelvű felületre barátságosan kezelhető osztályrendszert kialakítani az illesztés bonyolultabb részleteinek eltakarásával.

Az adatbázis-kezelés alapja – az ADO és más rendszerekhez hasonlóan – a `connect` és a `recordset` osztályok használatára épül. Egy kapcsolatobjektum (`session` vagy `connect`) létrehozása a következő függvényhívással valósítható meg:

```
// Egy œj kapcsolatobjektum lØtrehozÆsa
LOCAL conn // Ez lesz a kapcsolatobjektum
conn := ConnectNew("PG","localhost", "5432",
➥"postgres","111111","imiadatok")
```

Az osztálylétrehozónak átadott értékek jelentése:

- A `PG` azt jelzi, hogy egy PostgreSQL adatbázis-kapcsolatot szeretnénk.
- A második kapcsoló az adatbázis-kezelő gép IP-címe vagy DNS neve.
- A harmadik helyen adjuk meg a használni kívánt kaput.
- A negyedik és ötödik érték a felhasználónév és a jelszó.
- Utolsóként az adatbázis neve áll.

A `conn` objektumot a `conn:Destroy()` hívással szüntethetjük meg. A `conn` objektumon keresztül kezelhetjük az adatbázissal fenntartott munkafolyamatot, segítségével

Egy kedves olvasónk leveléből

A fentiekkel összhangban a Clippert (illetve a saját változatunkat, a CCC-t) nem régi programok feltámasztására használjuk, hanem újak írására. A CCC különösen alkalmas kiszolgálóprogramokhoz. Egy kiragadott példa, amire mostanában szakmai berkekben elismerően szoktak hümmögni: a CCC-ben készült banki számlavezető rendszerünkre épülő (szintén CCC-ben írt) XMLRPC-kiszolgálók nyújtanak tranzakciós felületet az Electra elektronikus banki rendszernek. Ugyanezek az XMLRPC-kiszolgálók biztosítanak lekérdezési felületet az Apache PHP modulja által futtatott ügyfélprogramoknak. Ha másik cikket nem is érdemes írni, annak örülnék, ha a lapban valahol megjelenne az következő hivatkozás: ➲ http://www.comfirm.hu. Itt sok leírás található a CCC-ről, maga a program is letölthető, bárki kedvére használhatja.

Vermes Mátyás

3. lista A párhuzamos példaprogramok megjegyzésekkel

```
// A pÆrhuzamos programozÆs bemutatÆsa.
local id1, id2
// Ezek tÆroljÆk a feladatazonos t kat.

id1:=start("f1")  // Ind tjuk az f1 f ggvØnyt.
id2:=start("f2")  // Ind tjuk az f2 f ggvØnyt.

// Ez a for ciklus a program fıszÆla.
for i=1 to 10

    ? 'main: send to ', id1, i,
➥TaskSendMsg(id1, i) // Az i ØrtØkØt
                      // elk ldj k
    ? 'main: send to ', id2, i,
➥TaskSendMsg(id2, i) // mindkØt szÆlnak

    sleep(1) // lemondunk a vezØrlØsrıl
             (az  nzı viselkedØs elker lØse)
next i
?
return
// Itt van vØge a fıszÆlnak Øs egyben
// a programnak is.

// Az elsı szÆl.
function f1
local m

while .t.
    m:=TaskGetMsg() // VÆr, am g meg nem
// kapja (blokkol dik a h vÆs),
    ? 'f1: got', m
// Amikor az adatot megkapja, ki rja.
end

// Ez a mÆsik szÆl algoritmusa.
function f2
local m

while .t.
    // Nem blokkol, de ha van ØrtØk,
    // kiolvassa, majd a kØpernyıre  rja
    m:=TaskPeekMsg()
    ? 'f2: got', m
    // NØha le kell mondani a processzorr l,
    // hiszen aTaskGetMsg() aszinkron f ggvØny.
    sleep(0.5)

end
```

4. lista Kép létrehozása

```
// KØplØtrehozÆs
//
#define PI    3.14159265358979323846
#include "gdinfo.ch"

local gd, red, white, black, blue, yellow, fon
clear screen

im=GdImageNew(200, 200)
// Egy  res vÆszon lØtrehozÆsa
white=im:newColor(255, 255, 255)
// a hasznÆlni k vÆnt sz nek
black=im:newColor()
red=im:newColor(255) // piros
blue=im:newColor(,,255) // kØk
yellow=im:newColor(255, 255) // sÆrga
fon=im:newColor(0xDE,0xEF,0xF5)
// a clip ezt az  rÆsm dot is ismeri
im:fill(0, 0, fon)
im:filledArc(100, 100, 45, 45, yellow)

luch=GdImageNew(3, 3)
luch:fill(0, 0, luch:newColor(255, 255))

im:setBrush(luch)
cx =100
cy = 100
// A napocska sugarainak kirajzolÆsa
for i=0 to 180 step 45
      x1 = cos(PI*i/180) * 60 + cx
      y1 = sin(PI*i/180) * 60 + cy
      x2 = cos(PI*(i+180)/180) * 60 + cx
      y2 = sin(PI*(i+180)/180) * 60 + cy
      im:line(x1, y1, x2, y2,
      ➥GDSTYLE_BRUSHED)
next

// A szemek
im:filledArc(93, 90, 6, 6, blue)
im:filledArc(107, 90, 6, 6, blue)

im:string("Ez a jelem!", 50, 50,
➥GDFONT_LARGE, black)
im:stringVector("", 50, 150, 300, 0, black)
// A megadott fÆjlba mentj k a kØpet
// (PNG t pus)
im:toPNG("kepem.png")
return .t.
```

tetszőleges SQL-parancsot adhatunk ki. A telefonok táblához például az alábbi módon hozhatunk létre rekordkészletet:

```
/ A select... parancs hasznÆlata
LOCAL rs
LOCAL sql := "select * from telefonok"
rs := conn:CreateRowset(sql)
➥// LØtrehozza a rekordkØszlet objektumot
```

Ezek után a rekordkészlet a szokásos módon kezelhető:

```
// VØgiglØpdel nk az SQL-mutat  (cursor)
// seg tsØgØvel Øs ki rjuk az elsı oszlopot
local data
DO WHILE !rs:Eof()
   data := rs:Read()
   // Az eredmØnytÆbla elsı oszlopÆnak
   //kØpernyıre listÆzÆsa
    ?
PADR(data[HASHSTR(UPPER(rs:FieldName(1)))])
   rs:Skip()
ENDDO
rs:Destroy()
```

Rugalmas lehetőség a `conn:Command(sql)` tagfüggvény, amellyel tetszőleges SQL-parancs (`insert`, `update`, `delete`, `commit`) adható ki:

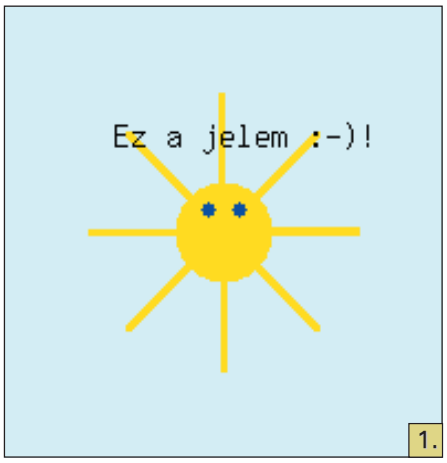
```
conn:Command("insert into telefonok
➥values('KovÆcs JÆnos', '123456789'")
conn:Command("commit")
```

A PostgreSQL-programozás elmélyítésére ajánlom a ***cliplibs/clip-postgresql/example*** könyvtárban található ***my_psql.prg*** programot, ami egy mini SQL-konzolt valósít meg clipben. A PostgreSQL modult használó programok fordításakor a clip parancssorban vagy a `Makefile`-ban a `-lclip-postgres` kapcsolót is meg kell adni.

Párhuzamos programozás

A clip lehetővé teszi, hogy a vele fejlesztett programok többszálúak legyenek. A forrásfájl könyvtárában e lehetőség megvalósítását a ***_thread.c*** fájl tartalmazza.

A clip csomagban néhány példaprogram is található, amelyek mutatják be az alaplehetőségeket. A *4. listában* látható kis program is ebből a gyűjteményből származik, elláttam azonban néhány magyarázattal. Az ottani magyarázatok, remélem, elégségesek a program működésének megértéséhez. Az `f1` függvény azért nem lesz önző, mert néha blokkolódhat. Ez az `f2`-re nem igaz, hiszen a `TaskPeekMsg` aszinkron végrehajtású, így szükség van a szál időnkénti közvetlen altatására (sleep). A fenti példa a szálak közti kapcsolattartásra egy egyszerű megoldást is bemutat.



1.

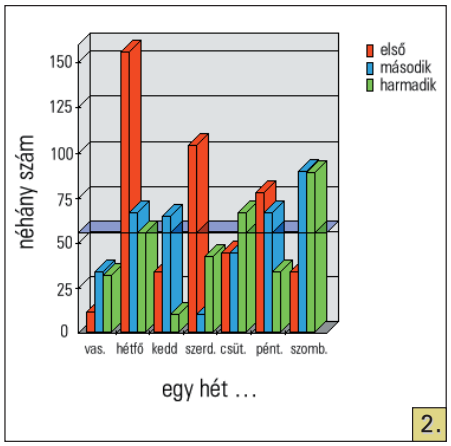
Hálózatos alkalmazások készítése

A clip rendszer egy alapszintű hálózatkezelő részt tartalmaz. Megfigyelésem szerint ez a modul még jelentős fejlesztésre szorul, de külső C/C++ eljárások használatával már most is teljes értékű programokat írhatunk. Nézzünk egy egyszerű clipprogramot, ami kapcsolatba lép egy webkiszolgálóval és egy érték nélküli `GET` kérést küld neki, majd

a kapott választ a képernyőre írja:

```
// httpteszt.prg
procedure main( ipcim )
 buff="GET  "+chr(10)+chr(13)
 eredmeny=space(1000)
 sock = tcpconnect( ipcim, 80 )
// Kapcsol dÆs a 80-as kapura
 tcpwrite(sock, @buff, len(buff))
// Adatk ldØs a TCP-foglalaton kereszt l
 tcpread(sock, @eredmeny, 1000 )
// A vÆlasz olvasÆsa
 tcpclose(sock)
// A TCP-foglalat lezÆrÆsa
 ? eredmeny
 ?
return
```

A programot a `clip -e httpteszt.prg` paranccsal kell lefordítani. A `-M` kapcsolót azért hagyhattuk el, mert `main` eljárást elkészítettünk, azaz nem kérjük annak önműködő létrehozását. A `./httpteszt 127.0.0.1` parancs például a helyi gép nyitóoldalát kéri le.



A Clip üzleti grafikája

A TCP/IP-foglalat (socket) illesztés a *_tcp.c* forrásfájlban lett megvalósítva. Érdekes lehet a következő, clipbeni osztályként megvalósított modulok forráskódjának tanulmányozása: SMTP-protokoll, HTML-, illetve CGI-támogatás (elemző és HTML-létrehozó lehetőségek), URL-osztály, aminek a használata hasonló a Java `URL class`-éhoz.

Titkosítás

Számos esetben szükségünk lehet rá, hogy értékes adatainkat titkosítsuk. A ***cliplibs/clip-crypto*** könyvtár `crypto.c` modulja remek megoldást kínál erre, egyúttal újabb példát láthatunk külső C-függvények illesztésére. Nézzük meg, hogyan is használhatjuk ezt az eszközt rendszerünkben!

```
// titok.prg
msg:='Kedves BarÆtom!'
// Ez a ny lt sz veg, amit titkos tani akarunk.
key='asdf'
// A titkos tÆshoz hasznÆlt kulcs.

msg1:=evp_encrypt(msg, key)
// msg1 a titkos tott sz veget tartalmazza
? msg1
// ki rjuk a kØpernyıre
msg2:=evp_decrypt(msg1, key)
// visszakapjuk az eredeti sz veget
? msg2
// ki rjuk a kØpernyıre.
?
```

A fordítás és összeszerkesztés a `clip -e -M titok.prg` segítségével lehetséges.

A gzip és bzip2 használata

Tegyük fel, hogy a ***csomag.gz*** fájl egy `gzip`-pel tömörített fájl. Ezt a `h:=gzipOpen("csomag.gz", "rwb")` függvénnyel megnyithatjuk, amelyben a `h` a fájlkezelő. A fájlt `gzipWrite(h, msg)` és `gzipRead(h, @msg1)` függvényekkel írhatjuk, olvashatjuk. A fájlkezelő műveletek befejezéseként tömörített fájlunkat a `gzipClose(h)` hívással zárhatjuk le.
A bzip2 formátumra is hasonló eljárások léteznek. Akit részletesebben is érdekel, nézze át a ***clip-bzip2*** és a ***clip-gzip*** könyvtárak tartalmát!

Dinamikus képlétrehozás

Webes alkalmazásokban gyakori lehetőség egy kép háttérben való létrehozása, így minden laplekérés alkalmával annak időszerűsített változatát láthatjuk. Ilyen eset például az, amikor a feltett kérdésre küldött ***Igen***, ***Nem***, ***Nem tudom*** válaszokat számoljuk, majd ezt az eloszlást egy oszlopdiagramban meg szeretnénk jeleníteni. A *4. listán* lévő látható példaprogram egy napocska ábráját (kislányom óvodai jele) hozza létre, feltüntetve benne az „Ez a jelem!” szöveget is. A program által létrehozott képet a *1. ábra* mutatja.
A program `im:toPNG()` sora helyett más képformátumokat is használhatunk (például `toJPEG()`).

További lehetőségek

A clip számos további, a fentiekben nem részletezett szolgáltatással bír. Ezek közül nagyon fontosak a grafikus felhasználói felület létrehozását, az üzleti grafikát (lásd *2. ábra*), a DBF fájlok SQL-értelmezőjét (interpreter) és a szabványos kifejezéseket kezelő (forrása ***_regex.c***) alrendszerek.
Befejezésül fontos kiemelni, hogy ez a rendszer folyamatos fejlesztés alatt áll, a fejlesztők honlapját néhány hetente érdemes megtekinteni, mert könnyen lehet, hogy a programozók valami új és érdekes modult illesztettek hozzá a már most is gazdag lehetőségeket nyújtó környezethez.

A cikkhez tartozó listák megtalálhatóak a 38. CD Magazin/Clipper könyvtárában.

Irodalomjegyzék

1. A clip hálózaton található leírása
   ➲ http://www.english.itk.ru/clipper/
2. ***Endrődi Tamás***: Korszerű programozás a Clipper 5.01-ben és a CA Clipperben, 1993
   ➲ http://www.gdf.hu/IAI/Segedletek/Segedlet009Fo.htm
3. ***Dr. Dedinszky Ferenc***: Clipper 5 ComputerBooks, 1992
4. ***Molnár Lajos-Molnárné Nagy Anikó***: Clipper kézikönyv Novotrade, 1989

Nyíri Imre
(inyiri@mol.hu) jelenleg a MOL Rt.-nél dolgozik. Informatikai vállalkozásában az Internet, a Linux, valamint a Java programozás gyakorlati hasznosításával foglalkozik, örök szerelme a C++ maradt.