Grafikus programok készítése GTK+-szal

Az egyik legmodernebb grafikus eszközkészlet (toolkit) a GTK+ nevű könyvtár, amelyet eredetileg a jól ismert Gimp rajzolóprogrammal együtt kezdtek el kifejleszteni. A neve, „The Gimp Toolkit”, is innen jön. Ma már rengeteg alkalmazás használja, és a Gnome is teljesen erre épül. Ebben a cikkben ennek a programozását mutatom be, részletezve a kezelésének alapjait, az eseményvezérelt programozás módját, és a konténerek használatát.

Bevezetés

Toolkitnek nevezzük azt a programkönyvtárat, amellyel egyszerűen rajzolhatunk felhasználói elemeket (***widget***), gombokat, menüket, gördítősávokat a programunkban. Az ***Xwindow*** rendszernél, a ***Windowszal*** ellentétben, ez nem része az alaprendszernek. Nem érdemes nekiállnunk az ***X*** rendszert ***toolkit*** nélkül, alacsony szinten programoznunk, ilyenkor ugyanis minden grafikát a legegyszerűbb elemekből, pontokból, vonalakból kell felépítenünk, és egyesével feldolgoznunk a szervertől jövő üzeneteket – egérmozdulatokat, kattintásokat, billentyű lenyomásokat. Az említett ***toolkitek*** többé-kevésbé leveszik a vállunkról ezt a feladatot. Az elavultabbakkal szerencsére a mai felhasználók már nem is találkoznak. Ezek meglehetősen csúnyák és nehézkesek voltak. Egyszerűbb programoknál az ***Xaw***, összetettebbeknél a ***Motif*** volt használatos, amely ráadásul nem is szabad szoftver. Manapság a két leggyakrabban alkalmazott ***toolkit a KDE-ben*** használt ***Qt*** és a ***Gnome*** alapját képező ***GTK+***.

Ha szeretnénk egy adott rendszer alá programot fejleszteni, érdemes megtanulnunk az abban megszokott ***toolkit*** használatát. Léteznek megoldások az egyes készletek hordozhatóvá tételére, de ezek mind a teljesítmény rovására mennek. Valljuk be őszintén, hogy például az ***OpenOffice.org*** vagy a ***Mozilla Firefox*** is jóval lassabbak, mint a ***Windowson*** futó natív társaik, ráadásul nem is illeszkednek teljesen egyik rendszerbe sem.

Helló, világ!

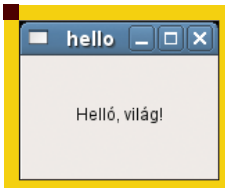
Először készítsünk egy programot, amely egy ablakot jelenít meg a képernyőn, benne a „Helló, világ!” felirattal! Gépeljük be az első listán látható forráskódot. Figyeljünk arra, hogy a szöveget UTF-8 kódolással mentsük el, a ***GTK+*** ugyanis belül mindig ezzel dolgozik. Ha a fájl neve ***hello.c***, akkor ezzel a ***Makefile*** részlettel fordítható:

```
hello: hello.c
    gcc $< -o $@ `pkg-config
    ➥gtk+-2.0 -cflags -libs`
```

Nézzük sorról sorra, mit csinál a program! Először is meghívjuk a `gtk_init()` függvényt. Ez átnézi a programunk argumentumlistáját, és amely paramétereket értelmezni tud (például `--display`), azt el is távolítja a listából, így a programunknak nem kell azokat figyelmen kívül hagynia. Ezután létrehozunk egy ablakot. A `gtk_window_new()` visszatérési értékét elmentjük egy változóba, hogy később hivatkozni tudjunk erre az ablakra. A következő sorban létrehozunk egy címkét (a `GtkLabel` olyan elem, amely csak egy szöveget ír ki). A `gtk_container_add()` függvénnyel el is helyezzük az ablakunkban. Ennek első paramétere az ablak, a második pedig a felületi elem, amelyet belerakjuk: itt az újonnan készített címke. Az első paraméter `GtkContainer*` típusú kell legyen, ezért a `GTK_CONTAINER()` makróval konvertáljuk. (Erre nincs feltétlenül szükség, csak a fordító figyelmeztető üzenetét kerüljük el vele. Írhattuk volna azt is, hogy `(GtkContainer *)window`. A makró azonban típusellenőrzést is végez.) Az ezt követő sorban a szöveg

1. Lista: Helló, világ GTK+ módra

```
#include <gtk/gtk.h>

int
main (int argc, char *argv[])
{
    GtkWidget *window;

    gtk_init (&argc, &argv);

    window = gtk_window_new
➥(GTK_WINDOW_TOPLEVEL);
    gtk_container_add
➥(GTK_CONTAINER (window),
➥gtk_label_new ("Helló,
➥világ!"));
    gtk_container_set_border_
➥width (GTK_CONTAINER
➥(window), 36);

    gtk_widget_show_all
➥(window);
    gtk_main ();
    return 0;
}
```



1. ábra GTK+ Helló, világ!

körüli keretet 36 képpontra állítjuk. A kód utolsó részében megjelenítjük az ablakot, és az azt tartalmazó összes elemet a `gtk_widget_show_all()` függvénnyel. A `gtk_label_new()` függvény visszatérési értékét, amely egy `GtkLabel*` típusú mutató, ezért nem kellett elmentenünk külön változóba. Mi később már nem hivatkozunk rá; a *GTK+* pedig számon tartja az ablak tartalmát. A függvény hatására az egész tartalmat bejárja, minden elemet, láthatóvá téve. Végül átadjuk a vezérlést a `gtk_main()` hívással.
Fordítsuk le a programot (`make`) és indítsuk el (`./hello`)! Megjelenik az 1. ábrán látható ablak. Az első programunk nem tartalmaz aktív elemeket, így nem tudunk sehogyan kilépni belőle. Észrevehetjük azt is, hogy ha az ablak jobb felső sarkában az X-re kattintunk, akkor is csak eltűnik, de a programunk fut tovább. (A terminál ablakban Crtl-C gombbal tudjuk megállítani.)

Aktív elemek, események, szignálok

A *GTK+* használatához meg kell értenünk néhány fogalmat.
Egy grafikus rendszerre írt program alapvetően más felépítésű, mint a konzolos társa. Az ilyet ugyanis ***eseményvezérelt*** (event-driven) módon kell megírnunk. A programnak a bejövő eseményeket kell feldolgoznia, és ha végzett, várni a továbbiakra. Az események lehetnek sokfélék, például lejár egy időzítő, a felhasználó megmozdítja az egeret, egy gombra kattint, vagy adat érkezett a hálózati kapcsolaton. A *GTK+* támogatást nyújt ilyen programok készítéséhez. A beérkező eseményeket maga kezeli: összegyűjti őket az úgynevezett ***event pool***-ban, és a megfelelő sorrendben végrehajtja az általunk hozzájuk rendelt programrészleteket.
Számunkra két fajta esemény fontos, amelyek között apró különbség van csak. ***Event*** a neve annak az eseménynek, amelyet az X szerver továbbít az alkalmazásunk felé. Ilyen lehet egy billentyű megnyomása, vagy egy kattintás valahol. A ***szignálok*** pedig általában már az ***eventeket*** értelmezik. Emiatt gyakran mindkettőre pongyolán szignálként szoktak hivatkozni. Ezek valamilyen objektum működésével kapcsolatos jelzések. Például egy gördítősávot mozgathatunk fogd és vidd módszerrel. Rákattintunk a csúszkára (***button-press-event***), arrébb húzzuk (az X szerver akár képpontonként küld egy ***motion-eventet***), és végül elengedjük (***button-release-event***). A felhasználói elemek szerencsére magasabb szintű szignálokat biztosítanak, például egy *GTK+* gördítősáv a ***value-changed*** jelzést küldi nekünk (emit), ha az értéke bármilyen módon megváltozott. Szignált mi magunk is indíthatunk. Egyes esetekben erre is szükség lehet: ha egy gördítősávot elmozdítunk a programból, hogy felhívjuk valamire a felhasználó figyelmét.
Egészítsük ki a programunkat, hogy kezelje az eseményeket (*2. lista*)!
A program hasonló az előzőhöz, viszont most a címke helyett egy gombot hozunk létre „Klikk ide!” felirattal, és ezt helyezzük el az ablakban. Ha a felhasználó erre kattint, akkor ez az elem kibocsájt egy *„**clicked**”* szignált, amelyhez mi a `klikk_cb()` függvényt kapcsoljuk. Akárhányszor a felhasználó a gombra kattint (vagy a szóköz, enter billentyűk valamelyikét megnyomja), a *GTK+* elindítja nekünk ezt a kezelőt (***callback***). A kezelő tartalmaz egy értékét megtartó (***static***) változót, amelyik számolja, hogy hányszor aktiváltuk a gombot, és ennek alapján megváltoztatja a feliratát. (A gomb az ablakhoz hasonlóan egy konténer elem, amely egy további elemet tartalmazhat.
A `gtk_button_new_with_label()` függvény tulajdonképpen egy kényelmi funkció, amely létrehoz egy gombot, és benne elhelyez egy adott címkét is.) A kezelőnek nincsen visszatérési értéke (***void***).
Az ablak *„**destroy**”* szignáljához is kapcsolunk egy kezelőt, ez a `gtk_main_quit()`. Ez a függvény kilép a programból, hatására a vezérlés a `gtk_main()` utáni sornál folytatódik. Amikor az ablak jobb felső sarkán az x-re kattintunk, vagy ***Alt-F4***-gyel bezárjuk azt, a szignál akkor jön létre. Látható, hogy nem feltétlenül kell minden szignálhoz kezelőt írnunk, használhatunk egyet a néhány előre definiált közül is.
Nézzük meg kicsit pontosabban, mi is történik az ablak bezárásakor!

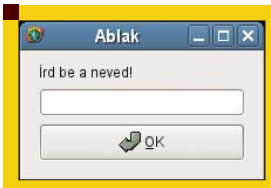
2. Lista: Eseményvezérelt Helló, világ

```
#include <gtk/gtk.h>

void
klikk_cb (GtkWidget * widget,
➥gpointer data)
{
   static int klikk = 0;
   char *text;

   text = g_strdup_printf
➥("Kattintások száma: %d",
➥++klikk);
   gtk_button_set_label
➥(GTK_BUTTON (widget),
➥text);
   g_free (text);
}

int
main (int argc, char *argv[])
{
   GtkWidget *window,
➥*button;

   gtk_init (&argc, &argv);

   window = gtk_window_new
➥(GTK_WINDOW_TOPLEVEL);
   gtk_container_set_border_
➥width (GTK_CONTAINER
➥(window), 36);
   g_signal_connect (G_OBJECT
➥(window), "destroy",
➥G_CALLBACK (gtk_main_quit),
➥NULL);

   button =
➥gtk_button_new_with_label
➥("Klikk ide!");
   gtk_container_add
➥(GTK_CONTAINER (window),
➥button);
   g_signal_connect (G_OBJECT
➥(button), "clicked",
➥G_CALLBACK (klikk_cb),
➥NULL);

   gtk_widget_show_all
➥(window);
   gtk_main ();
   return 0;
}
```

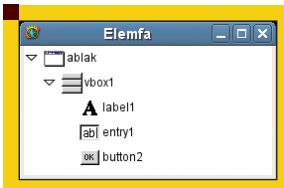
3. Lista A delete-event kezelője

```
gboolean
delete_event (GtkWidget *
➥widget, GdkEvent * event,
➥gpointer delete)
{
   GtkWidget *dialog =
➥gtk_message_dialog_new
➥(GTK_WINDOW (widget), 0,
      GTK_MESSAGE_QUESTION,
➥GTK_BUTTONS_YES_NO,
➥"Biztosan kilépsz?");
   int result;

   result = gtk_dialog_run
➥(GTK_DIALOG (dialog));
   gtk_widget_destroy
➥(dialog);
   if (result ==
➥GTK_RESPONSE_YES)
      return FALSE;

   return TRUE;
}
. . . . .
   gtk_container_set_border_
➥width (GTK_CONTAINER
➥(window), 36);
   g_signal_connect (G_OBJECT
➥(window), "delete-event",
➥G_CALLBACK (delete_event),
➥NULL);
   g_signal_connect (G_OBJECT
➥(window), "destroy",
➥G_CALLBACK (gtk_main_quit),
➥NULL);
```

Az X rendszer küld az alkalmazásunknak egy *„delete-event”* jelzést. Mivel ehhez mi nem adtunk meg kezelőt, a *GTK+* átadja az eredeti kezelőnek a vezérlést, amely bezárja az ablakot, az összes benne lévő elemmel együtt, és felszabadítja a hozzájuk tartozó erőforrásokat. A bezáráskor létrejön a *„destroy”* szignál, amelyhez viszont már definiáltunk egy kezelőt, amely pedig kilép a programból.
Az *eventek* kezelői, a szignálokéitól eltérően rendelkeznek visszatérési értékkel, mégpedig egy *gboolean* típusúval. Ha írunk egy ilyen függvényt, ezzel adhatjuk meg, hogy elvégeztük-e az eseményhez tartozó műveleteket, vagy nem. Ez nagyon hasznos dolog. Például készíthetünk a *„delete-event”* jelzéshez egy kezelőt, amelynek a hatására felugrik egy egyszerű dialógus, benne egy kérdéssel: „Biztos ki akarsz lépni a programból?”, alatta pedig Igen-Nem gombokkal. Ha a felhasználó a nemre kattint, akkor *TRUE* értékkel térünk vissza a kezelőből, jelezvén a *GTK+*-nak, hogy nincs további teendő. Ha pedig az igenre, akkor *FALSE* értékkel, aminek hatására a vezérlés az alapértelmezés szerinti kezelőre adódik tovább (*propagate the event further*), amely bezárja az ablakot. Az ablak bezárása pedig egy *„destroy”* szignált generál, amelyik kilép a programból.
A program kiegészítését lásd a *3. listán*. Ebben a `gtk_message_dialog_new()` egy olyan kényelmi funkció, amelyik rengeteg feladatot elvégez: létrehoz egy ablakot, rajzot, gombokat, szöveget tesz bele helyettünk.
A `gtk_dialog_run()` függvény visszatérési értéke adja meg, hogy melyik gombra kattintott a felhasználó. A dialógus ablak bezárása után döntjük el, hogy az alkalmazás ablakát engedjük-e bezárni az eredeti kezelőnek (`return FALSE`), vagy sem (`return TRUE`).



2. ábra A GTK+ konténerek



3. ábra Az ablak elemfája

A GTK+ tárolóelemek

A *GTK+-ban* vannak olyan elemek, amelyek továbbiakat tartalmazhatnak. Ilyen például a fenti gomb, amely egy címke elemet foglal magába. Lehetséges az is, hogy egy ikont és mellette egy szöveget mutat. Az ilyen elemeket nevezzük konténereknek. Aki készített már felhasználói felületeket *Windowsban*, tudja, hogy ott minden elemnek meg kell határoznunk a koordinátáját és a méretét, ahol az ablakon belül látszik. Ezzel ellentétben itt minden elem helyét és méretét az jelöli ki, hogy milyen konténerben helyeztük el és hogyan. Az elhelyezést akár egy fa gráfban, az elemfában le is rajzolhatjuk.
A konténer általános fogalom, rengeteg fajtája van. Vannak konténerek, amelyek egyetlen elemet tartalmaznak. Ilyen például egy ablak vagy egy gomb. Vannak olyanok is, amelyeknek több gyermekük (*child*) lehet, és azok között a rendelkezésre álló helyet az általunk meghatározott módon elosztják. Ilyenek a dobozok (vízszintes és függőleges, *GtkHBox*, *GtkVBox*) és a táblázat (*GtkTable*). A konténereket tetszés szerint egymásba ágyazhatjuk. A koncepciót a *2.* és *3. ábra* mutatja be. A legfelső szintű elem az ablak. Az tartalmaz egy függőleges dobozt, amely három részre osztja a területét. Az első elem egy címke, a középső egy szövegbeviteli doboz, a legalsó pedig a gomb. Az utóbbi is egyébként különálló gyermekekként foglalja magába a képet és az *OK* feliratot.
A *GTK+-ban* a felhasználói elemeket a `gtk_button_new()` és hasonló függvényekkel hozzuk létre, aztán egy konténerbe csomagoljuk (pack) őket. Léteznek kényelmi függvények is, amelyek több elemet hoznak létre egyszerre. Ilyen a már fent bemutatott `gtk_button_new_with_label()` is. Ez létrehoz egy gombot és egy címkét, majd a címkét elhelyezi a gomb (mint konténer) belsejében.
Több elem is van, amely képes konténerként működni. A *GTK+-ban* az egyes elemeket egymásból származtatják. Sok helyen ezért van szükség a típuskonverziós makrókra: a `gtk_container_add()` függvény, ahogy fent is láttuk, `GtkContainer*` típusú első argumentumot vár, de mi adhatunk neki egy gombot vagy egy ablakot is. Ezeket mind a `GtkContainer` típus leszármazottjai.
Módosítsuk programunkat úgy, hogy több gombot tartalmazzon (*4. lista*)!
A program sokban hasonlít az előzőre. A gombok létrehozása előtt kérünk egy *GtkVBox*-ot, amely az ablak belsejét függőlegesen több részre osztja. Ezt a dobozt tesszük az ablakba, majd a gombokat pedig egymás után ebbe a dobozba.
Létrehozunk egy egész számokat tároló `klikk[]` tömböt, ebben tároljuk majd, melyik gombunkra hányszor kattintott a felhasználó. Egy ciklussal létrehozzuk a tíz gombot, és egymás alá csomagoljuk őket a dobozba

4. Lista: Elemek egymásba ágyazása

```
#include <gtk/gtk.h>

void
klikk_cb (GtkWidget *widget,
int *data)
{
  char *text = g_strdup_printf
("Kattintások száma: %d",
++(*data));
  gtk_button_set_label
(GTK_BUTTON (widget), text);
  g_free (text);
}

int
main (int argc, char *argv[])
{
  GtkWidget *window, *button,
*vbox;
  const int gombok = 10;
  int klikk[gombok];
  int i;

  gtk_init (&argc, &argv);

  window = gtk_window_new
(GTK_WINDOW_TOPLEVEL);
  g_signal_connect (G_OBJECT
(window), "destroy",
G_CALLBACK (gtk_main_quit),
NULL);

  vbox = gtk_vbox_new (FALSE,
6);
  gtk_container_add
(GTK_CONTAINER (window),
vbox);
  for (i = 0; i < gombok; i++)
{
     klikk[i] = 0;
     button =
gtk_button_new_with_label
("Ide még nem
kattintottak!");
     g_signal_connect
(G_OBJECT (button),
"clicked", G_CALLBACK
(klikk_cb), klikk + i);

gtk_box_pack_start_defaults
(GTK_BOX (vbox), button);
  }

  gtk_widget_show_all
(window);
  gtk_main ();
  return 0;
}
```

(`gtk_box_pack_start_defaults()`). A gombok mindegyike ugyanúgy működik, mint az előző programban: a feliratán számolja az eddigi kattintásokat. A *GTK+* lehetőséget ad arra, hogy több különböző elemhez ugyanazt a kezelő függvényt kössük, az ugyanis megkapja az elemre mutató pointert (`GtkWidget *widget`). A `g_signal_connect()` függvény utolsó paraméterével minden kezelőhöz társíthatunk egy további mutatót is, amelyet saját célunkra használhatunk. Itt ez a `klikk[]` tömb arra elemére mutat, amelyben az adott gomb aktiválásait számoljuk; ez a `klikk+i` mutató. Ezt a mutatót a `klikk_cb()` függvény deklarációban bármilyen típusúnak definiálhatjuk, ugyanis a mutatók mérete mindig egyforma. Figyeljük meg, hogy ebben a programban sem használtunk egyetlen globális változót sem!

Ezen a programon a konténerek működését is kipróbálhatjuk. Futás közben az ablakot szabadon átméretezhetjük; a *GTK+* ekkor automatikusan újraszámolja a gombok méretét, és kirajzolja őket.

Tanácsok a fejlesztéshez

A ➲ *http://gtk.org/* oldalon megtaláljuk a *GTK+* dokumentációját, illetve egy tankönyvet is a ➲ *http://gtk.org/tutorial/* címen. Ez ehhez a cikkhez hasonló bevezetést nyújt (angolul). Itt az összes függvény leírását megtaláljuk; a dokumentációt érdemes a saját gépünkre is telepíteni (pl. *gtk-doc* csomag). A *Gnome* rendszer tartalmaz egy *Devhelp* nevű programot, amely a telepített dokumentációk böngészését teszi lehetővé.

A *GTK+-hoz* jár egy példaprogram gyűjtemény is. Ezt több terjesztésben a *gtk-examples* csomag tartalmazza, és a `gtk-demo` paranccsal indítható. Ez sok elem kezeléséhez mutat példákat, és egyben a forráskódot is megnézhetjük. Én magam is jobban szeretem, ha kezembe adják a példaprogramokat, amelyek alapján felépíthetem a saját alkalmazásomat. A *GTK+* viszont kitűnő minta arra, hogyan kellene felépíteni a programjainkat. Az egész könyvtár egy meglepően konzisztens, átlátható rendszert alkot; ha az ember kezd belejönni a használatába, már szinte automatikusan kitalálja a függvények neveit, amelyekkel az egyes elemeket kezelni lehet. Például egy beviteli dobozból a szöveget a `gtk_entry_get_text()` függvénnyel kérdezhetjük le, egy ablak címkéjét pedig a `gtk_window_set_title()` függvénnyel állíthatjuk be. A *Devhelp* programban az egyes függvényeket és szignálokat áttekintve percek alatt képet kaphatunk egy adott elem működéséről. Ha valamilyen műveletet nem találunk, amelyről úgy gondoljuk, kellene lennie, általában valamelyik szülő objektum rendelkezik olyannal: nincsen `gtk_window_show()`, de van `gtk_widget_show()`. Az utóbbi viszont bármilyen elemre alkalmazható.

Ha összetettebb felhasználói felületeket készítünk, akkor érdemesebb az egyes ablakokhoz létrehozni egy struktúrát, amelyben tároljuk a benne létrehozott elemeket (pontosabban azok `GtkWidget*` mutató reprezentációit). Ennek több előnye is van, azon kívül, hogy áttekinthetőbbé teszi a programunkat. Egy adott ablakból így könnyen több példányt készíthetünk. Gyakran előfordul az is, hogy a szignálok `user_data` paraméterében egy felületi elemet adunk át; az egész struktúrára mutató *pointer* átadásával minden elemre tud hivatkozni a kezelő.

Bonyolultabb felhasználói felületek létrehozásához mindenképpen érdemes a *Glade* programot kipróbálnunk, amely a ➲ *http://glade.gnome.org/* címen található meg. Ez a megtervezett ablakokat XML fájlokban is el tudja menteni, hogy futás közben töltsük be őket, de képes *C* forráskódot is generálni, amelyhez már csak a kezelőket kell megírnunk. Az áttekinthető felületek készítéséhez a *GNOME Human Interface Guidelines* dokumentum ad segítséget.

Ez a ➲ *http://developer.gnome.org/projects/gup/hig/2.0/* helyen található.

Czirkos Zoltán

Jelenleg diplomatervező a Budapesti Műszaki Egyetem Elektronikus Eszközök Tanszékén. Kutatási területe az operációs rendszerek betörésvédelme és a P2P kommunikáció. 2005-ben a Tudományos Diákköri Konferencián II. helyezést ért el. Kedvencei a boszorkányos és a rózsaszín párducos filmek.