Cours d'Interfaces Graphiques n°1

Edouard THIEL

Faculté des Sciences
Université d'Aix-Marseille (AMU)

Janvier 2016

Les transparents de ce cours sont téléchargeables ici :
http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ens/igra/

Lien court : http://j.mp/optigra

Présentation de l'UE

- Organisation :
  - 10 séances : 2h CM + 4h TP
  - Pas de contrôle continu, pas d'examen final
  - Évaluation : rendu TP + présentation de projet
  - Session de rattrapage : TP noté
- Contenu :
  - GTK+, Cairo
  - Programmation événementielle, animations, jeux 2D
  - Courbes de Bézier, Transformations géométriques
- Sur la page web du cours : transparents, planches, liens, etc.

Plan du cours n°1

1. Le projet GTK+
2. Installation
3. Compilation d'un programme
4. Hello World!
5. User data
6. Premier bouton

1 - Le projet GTK+

Acronymes : GTK+ = GIMP ToolKit
GIMP = GNU Image Manipulation Program
GNU = GNU's Not Unix

GIMP : 1995, en C, multi-plateformes, GPL v3+
projet 2 étudiants UC Berkeley
traitement d'images et photocomposition

GTK+ : en C, multi-plateformes, LGPL v2.1
couche graphique : fenêtres, boutons, menus, etc
originalement en remplacement de Motif
devient un projet autonome en 1998

Que signifie le +

GTK+ écrit en C objet :

- chaque élément graphique (fenêtre, bouton, menu) est un *objet*
- Un tel objet sait s'afficher, se dimensionner, se créer, se détruire, etc
- Dérivation : une case à cocher à partir d'un bouton
- *widget* = Window GadgET

Fait à l'aide de macros.

Binding de langages

Les librairies GTK+ sont accessibles dans d'autres langages :

C++	gtkmm
C#	Gtk#
D	GtkD
Haskell	Gtk2Hs
Java	java-gnome
Javascript	Gjs, Seed
Python	PyGTK
...	

Environnements de bureaux

bureau, fenêtres, icones, barres, filemgr, D&D, Bus, plugins, etc

Nombreux à être programmés en GTK+ :

GNOME	GNOME Shell, Unity, ...
MATE	fork de GNOME 2
LXDE, Xfce	légers
Cinnamon	Linux MINT
...	

Multi-plateforme

Sur Unix	X11, Wayland, DirectFB (léger), ncurses (console)
Sur MacOS	XQuartz (X11.app), Quartz (natif)
Sur Windows	API Win32
Dans navigateur	Broadway (HTML5 et web sockets)

Concurrence

Nombreux toolkits multi-plateforme :

Qt	C++, bureau KDE
wxWidgets	C++, 1992 - Audacity, BitTorrent, Code::Blocs, ...
EFL	Enlightenment, C - freebox, certains Samsung, ...
Tk	C, 1991 - interprété
XUL	XML - firefox
AWT, Swing	java
...	

Notes : sur Linux, wxWidgets s'appuie sur GTK+
KDE est compatible GTK+

Versions

Un développement intensif, qui ne ralentit pas :

1.0	1998	première version stable
2.0	2002	GObject, UTF-8
2.8	2005	Cairo
2.24	2011	dernière 2.x
3.0	2011	réorganisation librairies
3.8	2013	support Wayland 1.0
3.19	2016	actuelle

2 - Installation sur Ubuntu

```
$ sudo apt-get install libgtk-3-dev libgtk-3-doc
...
Les paquets supplémentaires suivants seront installés :
  libatk-bridge2.0-dev libatk1.0-doc libxkbcommon-dev
Paquets suggérés :
  libgtk2.0-doc devhelp
Les NOUVEAUX paquets suivants seront installés :
  libatk-bridge2.0-dev libatk1.0-doc libgtk-3-dev libgtk-3-doc
  libxkbcommon-dev
...
Paramétrage de libatk-bridge2.0-dev:amd64 (2.10.2-2ubuntu1) ...
Paramétrage de libatk1.0-doc (2.10.0-2ubuntu2) ...
Paramétrage de libxkbcommon-dev (0.4.1-0ubuntu1) ...
Paramétrage de libgtk-3-dev (3.10.8-0ubuntu1.6) ...
Paramétrage de libgtk-3-doc (3.10.8-0ubuntu1.6) ...
```

Documentation

Paquet libgtk-3-doc : doc html en local

`file:///usr/share/doc/libgtk-3-doc/gtk3/index.html`

Par exemple "Widget Gallery" :

`file:///usr/share/doc/libgtk-3-doc/gtk3/ch03.html`

Devhelp

Autre source de documentation : devhelp

```
$ sudo apt-get install devhelp
...
Les paquets supplémentaires suivants seront installés :
  devhelp-common libdevhelp-3-2
Les NOUVEAUX paquets suivants seront installés :
  devhelp devhelp-common libdevhelp-3-2
...
Paramétrage de devhelp-common (3.8.2-2ubuntu1) ...
Paramétrage de libdevhelp-3-2 (3.8.2-2ubuntu1) ...
Paramétrage de devhelp (3.8.2-2ubuntu1) ...
Traitement déclenché pour  libc-bin (2.19-0ubuntu6.6) ...
Traitement déclenché pour  menu (2.1.46ubuntu1) ...
```

Taper devhelp puis gallery ...

3 - Compilation d'un programme écrit en GTK+

GTK+ nécessite de nombreuses librairies :

- elles dépendent de l'architecture ;
- les noms et chemins dépendent de la distribution ;
- nécessité d'outils pour les recenser.

Script `pkg-config` :

`--help`	affiche l'aide
`--cflags gtk+-3.0`	affiche les chemins des headers
`--libs gtk+-3.0`	affiche les librairies et chemins

Exemple

```
$ pkg-config --cflags gtk+-3.0
-pthread -I/usr/include/gtk-3.0 -I/usr/include/atk-1.0
-I/usr/include/at-spi2-atk/2.0 -I/usr/include/pango-1.0
-I/usr/include/gio-unix-2.0/ -I/usr/include/cairo
-I/usr/include/gdk-pixbuf-2.0 -I/usr/include/glib-2.0
-I/usr/lib/x86_64-linux-gnu/glib-2.0/include
-I/usr/include/harfbuzz -I/usr/include/freetype2
-I/usr/include/pixman-1 -I/usr/include/libpng12

$ pkg-config --libs gtk+-3.0
-lgtk-3 -lgdk-3 -latk-1.0 -lgio-2.0 -lpangocairo-1.0
-lgdk_pixbuf-2.0 -lcairo-gobject -lpango-1.0 -lcairo
-lgobject-2.0 -lglib-2.0
```

Compilation simple

Pour compiler un programme `p1.c` :

```
$ gcc -Wall -std=c99 $(pkg-config --cflags gtk+-3.0) \
    p1.c -o p1 $(pkg-config --libs gtk+-3.0)
```

Avec des modules `m1.c` et `m2.c` :

```
$ gcc -Wall -std=c99 $(pkg-config --cflags gtk+-3.0) \
    p1.c m1.c m2.c -o p1 $(pkg-config --libs gtk+-3.0)
```

Compilation séparée

On peut compiler séparément :

```
$ gcc -Wall -std=c99 $(pkg-config --cflags gtk+-3.0) -c p1.c
$ gcc -Wall -std=c99 $(pkg-config --cflags gtk+-3.0) -c m1.c
$ gcc -Wall -std=c99 $(pkg-config --cflags gtk+-3.0) -c m2.c
$ gcc p1.o m1.o m2.o -o p1  $(pkg-config --libs gtk+-3.0)
```

Avantages :

- compiler ce qui a changé
- moins d'erreurs à la fois

Automatisation

Différents outils en fonction de la complexité :

- Script
- Makefile
- Autotools

4 - Hello World!

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

int main (int argc, char *argv[])
{
  GtkApplication *app;
  int status;

  app = gtk_application_new (NULL, G_APPLICATION_FLAGS_NONE);
  g_signal_connect (app, "activate",
                    G_CALLBACK(on_app_activate), NULL);
  status = g_application_run (G_APPLICATION(app), argc, argv);
  g_object_unref (app);

  return status;
}
```

Signal

Évènement collecté par GTK (création, clic souris...) et transmis aux widgets.

Identifié par un mot : `activate`, `clicked`, ...

Pour recevoir un signal, un widget connecte une *callback* :

- fonction de votre programme
- signature imposée par GTK selon signal
- appelée par `g_application_run` à réception d'un signal

Pour connecter une callback :

```
g_signal_connect (app, "nom_du_signal",
                  G_CALLBACK(on_widget_signal), NULL);
```

Hello World! (suite)

Callback `on_app_activate`, appelée lors du signal `activate` pour créer les widgets :

```
void on_app_activate (GtkApplication* app, gpointer user_data)
{
  GtkWidget *window;

  window = gtk_application_window_new (app);
  gtk_window_set_title (GTK_WINDOW(window), "Hello World!");
  gtk_window_set_default_size (GTK_WINDOW(window), 400, 300);
  gtk_widget_show_all (window);
}
```

Compilation

```
$ gcc -Wall -std=c99 $(pkg-config --cflags gtk+-3.0) -c ex01-hello.c

$ gcc ex01-hello.o -o ex01-hello $(pkg-config --libs gtk+-3.0)

$ ./ex01-hello
```

5 - User data

Mécanisme pour éviter les variables globales : les "user data".

Principe :

- on crée un struct "fourre-tout" qui contiendra *toutes* nos données ;
- on associe ce struct aux signaux ;
- GTK nous le donne en `user_data` dans les callbacks.

Exemple :

```
typedef struct {
    GtkWidget *window;
    char *title;
} Mydata;
```

User data (suite)

```
int main (int argc, char *argv[])
{
  Mydata my;
  my.title = "Hello World Again!";
  ...
  g_signal_connect (app, "activate",
                    G_CALLBACK(on_app_activate), &my);
  ...
}

void on_app_activate (GtkApplication* app, gpointer user_data)
{
  Mydata *my = user_data;

  my->window = gtk_application_window_new (app);
  gtk_window_set_title (GTK_WINDOW(my->window), my->title);
  gtk_window_set_default_size (GTK_WINDOW(my->window), 400, 300);
  gtk_widget_show_all (my->window);
}
```

6 - Premier bouton

Ingrédients :

- Un conteneur de boutons : `gtk_button_box_new`
- Un bouton : `gtk_button_new_with_label`
- Rattachement : `gtk_container_add`

Premier bouton (suite)

```
void on_app_activate (GtkApplication* app, gpointer user_data)
{
  Mydata *my = user_data;
  GtkWidget *button_box, *button1;

  my->window = gtk_application_window_new (app);

  gtk_window_set_title (GTK_WINDOW (my->window), my->title);
  gtk_window_set_default_size (GTK_WINDOW (my->window), 400, 300);

  button_box = gtk_button_box_new (GTK_ORIENTATION_HORIZONTAL);
  gtk_container_add (GTK_CONTAINER (my->window), button_box);

  button1 = gtk_button_new_with_label ("Click me");
  gtk_container_add (GTK_CONTAINER (button_box), button1);

  gtk_widget_show_all (my->window);
}
```

Action au clic

Lorsque le bouton est cliqué, le signal `clicked` est émis
Il suffit d'attacher une callback au signal :

```
void on_button1_clicked (GtkWidget *widget, gpointer data)
{
  Mydata *my = data;
  printf ("Button clicked in window '%s'\n", my->title);
}

void on_app_activate (GtkApplication* app, gpointer user_data)
{
  ...
  g_signal_connect (button1, "clicked",
                    G_CALLBACK(on_button1_clicked), my);
  ...
}
```

Bouton quitter

```
void on_b_quit_clicked (GtkWidget *widget, gpointer data)
{
  Mydata *my = data;
  printf ("Closing window and app\n");
  gtk_widget_destroy(my->window);
}

void on_app_activate (GtkApplication* app, gpointer user_data)
{
  ...
  GtkWidget ..., *b_quit;
  ...
  b_quit = gtk_button_new_with_label ("Quit");
  gtk_container_add (GTK_CONTAINER (button_box), b_quit);
  g_signal_connect (b_quit, "clicked",
                    G_CALLBACK(on_b_quit_clicked), my);
  ...
}
```