TP GTK

Pierre Roux*

27 et 28 février 2012

1 Quelques conseils avant de commencer

Veiller à avoir installer GTK, sa documentation et glade. Sur Debian et dérivés comme Ubuntu, la commande suivante devrait suffire

```
# apt-get install glade libgtk2.0-doc  # en root ou précédé de sudo
```

La documentation est souvent indispensable lorsqu'on utilise GTK, si on a installé le paquet précédent, elle est disponible sur le système à l'adresse `file:///usr/share/doc/libgtk2.0-doc/gtk/index.html`.

Ce TP va beaucoup s'inspirer du tutoriel officiel `http://library.gnome.org/devel/gtk-tutorial/2.20/`.

GTK est écrit suivant le paradigme objet (et oui on peut faire de l'objet en C et pas seulement en C++ ou en JAVA) mais on ne s'attardera pas particulièrement sur ce point.

2 Notions de programmation événementielle

Jusqu'à maintenant, les programmes que nous avons écrit dans ce cours suivaient le schéma suivant :

1. lecture (scanf) ;
2. calcul ;
3. affichage du résultat (printf).

Il est difficile de reproduire ce schéma avec une interface graphique, puisque – sauf à se limiter à un unique bouton – l'utilisateur dispose de plusieurs moyens d'entrer des informations dans notre application. Il faut donc adapter le schéma précédent :

1. dessiner la fenêtre ;
2. tant que l'utilisateur ne souhaite pas quitter le logiciel ;
3. vérifier s'il a cliqué sur quelque chose, remplis un champ de texte,... (événement) ;
4. si oui, réaliser l'action associée à cet événement ;
5. recommencer en 2.

Cette boucle est généralement appelée boucle événementielle ou boucle principale (`mainloop` dans la langue de Shakespeare). Il serait pénible de devoir la réaliser nous même, d'autant plus que nous n'avons aucune envie de devoir gérer des détails sordides tels que le fait qu'un bouton doit changer d'apparence lors de l'événement « la souris passe dessus » ou que sais je encore. On laissera donc la bibliothèque se charger elle même de cette boucle. Il ne nous restera plus alors qu'à indiquer à la bibliothèque quelles sont les actions à réaliser pour chaque événement qui nous intéresse (c'est ce qu'on appellera `callback` par la suite) et lancer cette boucle principale. Mais place à la pratique...

*`pierre.roux@enseeiht.fr`

3 Notre premier programme graphique

3.1 Notre première fenêtre

Tapez le programme suivant dans un fichier `base.c` :

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

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

  gtk_init(&argc, &argv);  /* initialise la librairie,
                            * cette ligne est obligatoire
                            * au début de tout programme utilisant GTK. */

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);  /* crée une fenêtre */
  gtk_widget_show(window);  /* la rend effectivement visible */

  gtk_main();  /* et lance la boucle principale */

  return 0;
}
```

et compilez et exécutez le avec les commande suivante :

```
% gcc base.c -o base $(pkg-config --cflags --libs gtk+-2.0)
% ./base
```

vous devriez voir apparaître une fenêtre.

Que se passe t-il si on ferme cette fenêtre ? La fenêtre disparaît mais la boucle principale continue à tourner (le shell ne nous rend pas la main). Normal, nous n'avons pas spécifié d'action à réaliser pour l'événement « fermer la fenêtre ». On peut y remédier de la façon suivante :

```
/* fonction à rajouter avant le main */
static void destroy(GtkWidget *widget, gpointer data)
{
    gtk_main_quit ();  /* fonction qui met fin à la boucle principale */
}

/* appel à rajouter dans le main entre la création de la fenêtre et son affichage. */
  /* Définir quoi faire sur l'événement "destroy",
   * le premier argument est la fenêtre,
   * le deuxième designe l'événement,
   * le troisième est un pointeur de fonction vers la fonction précédente,
   * et le dernier est le pointeur, qui sera passé en deuxième argument
   * de cette fonction (ici on ne passe rien). */
  g_signal_connect(window, "destroy", G_CALLBACK (destroy), NULL);
```

Si on recompile, on constate que la fermeture de la fenêtre déclenche cette fois la fin du programme. C'est mieux mais notre fenêtre est toujours bien vide...

3.2 Remplissons la un peu

Ajoutons un bouton qui affichera « Bonjour tout le monde ! » (c'est adéquat vu l'horaire de ces TP) quand on cliquera dessus.

```
/* un deuxième callback pour que le premier se sente moins seul */
static void callback_click_bouton(GtkWidget *widget, gpointer data)
{
  printf("Bonjour tout le monde !\n");
}

/* puis dans le main */
  GtkWidget *bouton;

  /* on crée le bouton */
  bouton = gtk_button_new_with_label("Cliquez moi !");

  /* et on connecte le callback, comme le précédent */
  g_signal_connect(bouton, "clicked", G_CALLBACK(callback_click_bouton), NULL);

  /* on ajoute le bouton à la fenêtre */
  gtk_container_add(GTK_CONTAINER(window), bouton);
  /* et on l'affiche */
  gtk_widget_show(bouton);
```

4 Exercices

4.1 Passer des données aux callbacks

Tous les callbacks prennent un argument `data`. C'est un pointeur générique, le type `gpointer` étant défini quelque part par `typedef void* gpointer;` (c.f. file:///usr/share/doc/libglib2.0-doc/glib/glib-Basic-Types.html#gpointer si vous avez installé le paquet `libglib2.0-doc`). Ce pointeur permet de spécifier lors de la mise en place du callback des données qui seront passées au callback lors de son appel.

Écrire un callback `imprime_chaine` qui prendra comme argument `data` une chaîne de caractère et l'imprimera avec `printf("%d"...`.

Modifier l'installation du callback du bouton pour utiliser le nouveau.

4.2 Ajouter un deuxième bouton

Mettre un deuxième bouton qui utilisera le callback de l'exercice précédent (les deux boutons déclencheront donc le même callback) pour afficher une deuxième chaîne de caractère.

4.3 Les champs de texte

Jusqu'à maintenant nos jolies (ou pas) interfaces graphiques n'ont servi qu'à entrer des données, mais personne ne veut lire le résultat sur la sortie standard. Regarder la doc du widget (abréviation de window gadget) `label` (file:///home/share/doc/libgtk2.0-doc/gtk/GtkLabel.html) et voyez comment remplacer les printf par un message dans un tel champ de texte.