Projet de Programmation

Benoit Donnet

Année Académique 2023 - 2024

Agenda

Partie 3: Eléments de Programmation Evénementielle

- Chapitre 1: Introduction aux Interfaces Graphiques
- Chapitre 2: Applications Interactives
- Chapitre 3: Pattern MVC

Agenda

- Chapitre 1: Introduction aux Interfaces Graphiques
  - Programmation Evénementielle
  - GTK
  - Fenêtres
  - Labels
  - Boutons
  - Boxes
  - Menu
  - Tables

Agenda

- Chapitre 1: Introduction aux Interfaces Graphiques
  - Programmation Evénementielle
    - ✓ Principe
    - ✓ Gestion des Evénements
  - GTK
  - Fenêtres
  - Labels
  - Boutons
  - Boxes
  - Menu
  - Tables

Principe

- Programmation classique
  - l'application a le contrôle
  - l'utilisateur fait ce que l'application lui demande

```
#include <stdio.h>

int main(int argc, char **argv){
   const int N = 10000;
   char buf[N];

   //...

   printf("fichier: ");
   scanf("%s", buf);

   //...
   return 0;
}//fin programme
```

fichier:
toto.txt

Principe (2)

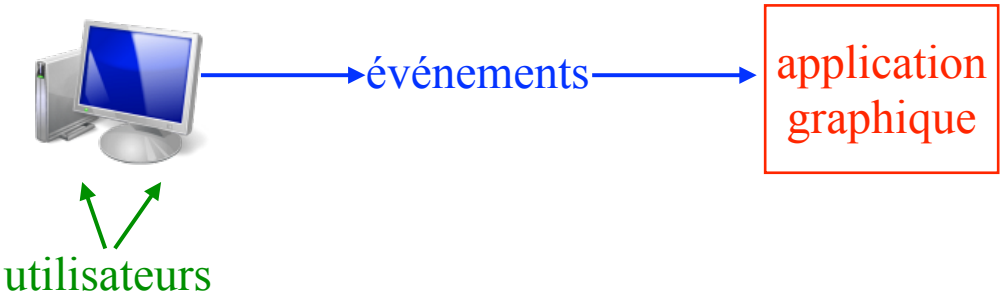
- Interface graphique
  - l'utilisateur a le contrôle
    - ✓ il peut, à tout moment, sélectionner, cliquer, changer de fenêtre, ...
  - l'application est "esclave" de l'utilisateur

```
void detruire(...){
  gtk_main_quit();
}

int main(int argc, char **argv){
   //...

   g_signal_connect(...,
        G_CALLBACK(detruire), ...);

   //...
   return 0;
}//fin programme
```

utilisateurs

Principe (3)

- Conséquences?
  - l'application doit toujours être prête à réagir
  - programmation **<u>événementielle</u>**
- **<u>Evénement</u>**
  - on parle aussi de message
  - envoyé à l'application ciblée
  - à chaque action de l'utilisateur



Principe (4)

- Exemples d'événements
  - appuyer/relâcher un bouton de la souris
  - appuyer/relâcher une touche du clavier
  - appuyer/glisser sur l'écran tactile
  - bouger la souris avec un bouton enfoncé
  - la souris entre dans/sort d'une fenêtre
  - rafraîchir la fenêtre
    - ✓ la fenêtre redevient visible
  - redimensionner la fenêtre
  - faire apparaître/disparaître la fenêtre de l'écran

Principe (5)

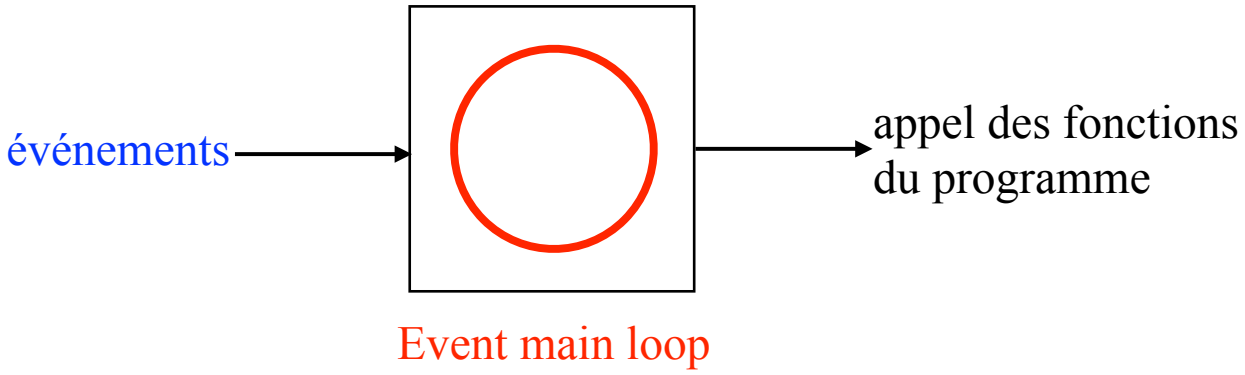
- Un événement
  - est un "objet" envoyé par l'application
    - via un **signal**
  - contient des informations dépendant du type d'événement
    - exemple: la position (x, y) de la souris
- Technique très générale
  - pas limitée aux interfaces graphiques
  - pour la communication entre objets, entre applications
  - exemples
    - communication d'applications via le réseau
    - signaux UNIX
    - ...

Gestion Evénements

- Comment gérer les événements?
  1. créer les principaux objets graphiques
  2. lancer la **boucle de gestion des événements**
- L'application se met en attente des événements
  - l'application n'appelle aucune fonction de son propre chef
  - elle devient esclave de l'utilisateur

Gestion Evénements (2)

- Boucle de gestion des événements (*event main loop*)
  - boucle infinie qui
    - ✓ récupère les événements
    - ✓ appelle les fonctions du programme



Gestion Evénements (3)

- Il existe deux façons d'implémenter la boucle de gestion des événements
  1. protocole **non embarqué**
     - ✓ le programmeur écrit lui-même cette boucle
     - ✓ à lui de prendre en compte
       - tous les objets graphiques
       - tous les événements utiles
       - toutes leurs combinaisons temporelles
     - ✓ difficilement gérable en pratique

Gestion Evénements (4)

- Il existe deux façons d'implémenter la boucle de gestion des événements (cont.)

  2. protocole **embarqué**
     - ✓ la boucle est gérée par le système, pour l'utilisateur
     - ✓ le contrôle du dialogue est embarqué dans les objets graphiques
     - ✓ détection des événements via des callbacks
     - ✓ utilisation
       - ‣ implémentation des callbacks
       - ‣ association des callback aux fenêtres
       - ‣ exécution automatique quand une condition se produit

Agenda

- Chapitre 1: Introduction aux Interfaces Graphiques
  - Programmation Evénementielle
  - GTK
    - ✓ Historique
    - ✓ Installation
    - ✓ Premier Programme
  - Fenêtres
  - Labels
  - Boutons
  - Boxes
  - Menu
  - Tables

Historique

- GTK+?
  - Gimp Toolkit
  - http://www.gtk.org/
- A l'origine, boîte à outils pour les développeurs du projet Gimp
  - *Gnu Image Manipulation Program*
- GTK+ se détache de Gimp en 1997
- Contient, entre autre, une bibliothèque permettant de créer des interfaces graphiques (GUI)

Installation

- Installation
  - http://www.gtk.org/download/index.php
- Compilation d'un projet GTK+
  - il faut ajouter l'option `` `pkg-config --cflags --libs gtk+-2.0` ``
  - à rajouter dans le Makefile

Premier Programme

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

int main(int argc, char **argv){
  gtk_init(&argc, &argv);

  //création des différents objets de la GUI

  gtk_main();

  return EXIT_SUCCESS;
}//fin programme
```

bibliothèque GTK+

initialisation de GTK+

lancement de la boucle des événements

défini dans stdlib.h

Agenda

- Chapitre 1: Introduction aux Interfaces Graphiques
  - Programmation Evénementielle
  - GTK
  - Fenêtres
    - ✓ Hiérarchie
    - ✓ Affichage
    - ✓ Destruction
    - ✓ Signaux
    - ✓ Personnalisation
  - Labels
  - Boutons
  - Boxes
  - Menu
  - Tables

Hiérarchie

- GTK+ est écrit en Orienté Objet
  - notion de hiérarchie/héritage
- Tous les objets graphiques héritent des propriétés et des fonctions d'une widget de base
  - `GtkWidget`
- Hiérarchie pour les fenêtres

```
GObject
  GtkObject
    GtkWidget
      GtkContainer
        GtkBin
          GtkWindow
```


Affichage

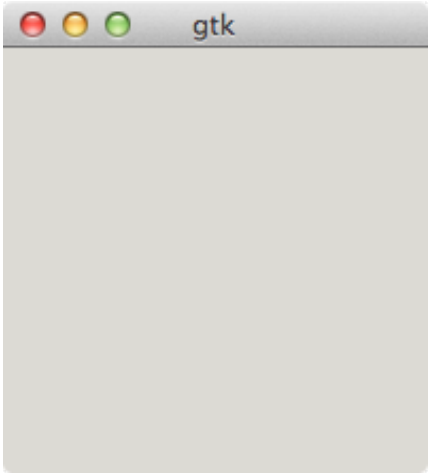
- On manipule un pointeur sur GtkWidget
  - `GtkWidget *pWindow;`
- La création d'une fenêtre se fait via la fonction
  - `GtkWidget *gtk_window_new(GtkWindowType)`
    - ✓ `GTK_WINDOW_TOPLEVEL`
    - ✓ `GTK_WINDOW_POPUP`
- On peut, enfin, afficher la fenêtre
  - `void gtk_widget_show(GtkWidget *)`

Affichage (2)

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

int main(int argc, char **argv){
  gtk_init(&argc, &argv);

  GtkWidget *pFenetre = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_widget_show(pFenetre);

  gtk_main();

  return EXIT_SUCCESS;
}//fin programme
```


Affichage (3)

```
$>gcc -o gtk gtk2.c `pkg-config --cflags --libs gtk+-2.0`
$>./gtk
```



Destruction

- On détruit une fenêtre en utilisant la fonction
  - `void gtk_widget_destroy(GtkWidget *)`
- Comment savoir qu'il faut tuer la fenêtre?
  - i.e., comment savoir que le programme est terminé?
- Quand l'utilisateur indique que le programme est terminé
  - comment le savoir?
    - ✓ signaux!
    - ✓ programmation événementielle

Signaux

- Lorsque l'utilisateur interagit avec l'application, le widget concerné émet un signal
- A chaque widget est associé un/plusieurs signal/ signaux
- Il est possible d'associer une action à un signal
- Quand un signal est émis, GTK vérifie la présence d'une action associée
  - si oui, alors l'action est exécutée
  - sinon, rien ne se passe
- La fonction qui implémente l'action est appelée fonction **callback**

Signaux (2)

- Comment créer une fonction callback?
  - Prototype général d'un callback

```
void id(GtkWidget *, gpointer)
```

gpointer : donnée supplémentaire

GtkWidget * : widget qui a émis le signal

- Comment associer un signal à une fonction callback?

```
gulong g_signal_connect(gpointer *,
                        const gchar *,
                        GCallback,
                        gpointer)
```

gpointer * : source du signal
const gchar * : signal à intercepter
GCallback : fonction callback
gpointer : donnée supplémentaire

Signaux (3)

```
#include <stdlib.h>
#include <gtk/gtk.h>
void detruire_fenetre(GtkWidget *pF, gpointer data){
  gtk_main_quit();
}//fin detruire_fenetre()

int main(int argc, char **argv){
  gtk_init(&argc, &argv);

  GtkWidget *pFenetre =
  gtk_window_new(GTK_WINDOW_TOPLEVEL);
  g_signal_connect(G_OBJECT(pFenetre), "destroy",
                   G_CALLBACK(detruire_fenetre), NULL);
  gtk_widget_show(pFenetre);

  gtk_main();

  return EXIT_SUCCESS;
}//fin programme
```

Personnalisation

- Il est possible de personnaliser une fenêtre
- Position de la fenêtre
  - `void gtk_window_set_position(GtkWindow *, GtkWindowPosition)`
  - `void gtk_window_move(GtkWindow *, int, int)`
  - `void gtk_window_get_position(GtkWindow *, gint *, gint *)`
- Titre de la fenêtre
  - `void gtk_window_set_title(GtkWindow *, const gchar *)`
- Taille de la fenêtre
  - `void gtk_window_set_default_size(GtkWindow *, gint, gint)`

Agenda

- Chapitre 1: Introduction aux Interfaces Graphiques
  - Programmation Evénementielle
  - GTK
  - Fenêtres
  - Labels
    - ✓ Hiérarchie
    - ✓ Création
    - ✓ Insertion de Texte
    - ✓ Affichage
  - Boutons
  - Boxes
  - Menu
  - Tables

Hiérarchie

- A l'instar des fenêtres, les labels obéissent à une hiérarchie dans GTK+

```
GObject
  GtkObject
    GtkWidget
      GtkMisc
        GtkLabel
```


Création

- On manipule un pointeur sur GtkWidget
  - `GtkWidget * pLabel;`
- On initialise l'objet
  - `GtkWidget *gtk_label_new(const char *)`

```
GTKWidget *pLabel = gtk_label_new("Back in black.  I hit
the sack!");
```

Insertion

- L'affichage du label nécessite son insertion dans la fenêtre principale
- Comment placer un widget dans un autre widget?
  - **container**
    - ✓ permet de contenir et afficher un autre widget
- Il faut distinguer
  - le widget conteneur
  - le widget contenu
- Une fenêtre est aussi un container
  - cfr. hiérarchie où `GtkWindow` hérite de `GtkContainer`

Insertion (2)

- Comment insérer le widget dans le widget conteneur?
  - `void gtk_container_add(GtkContainer *, GtkWidget *)`

```
gtk_container_add(GTK_CONTAINER(pWindow), pLabel);
```

Affichage

- `gtk_widget_show()` n'affiche que la fenêtre et pas son contenu
- Comment faire pour tout afficher?
  - `gtk_widget_show()` en cascade sur les différents éléments de la fenêtre
  - possible mais boarf
    - ✓ trop long
    - ✓ trop fastidieux, surtout pour les fenêtres complexes
- Solution
  - `void gtk_widget_show_all(GtkWidget *)`
  - affiche la fenêtre et tous ses composants

Affichage (2)

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

int main(int argc, char **argv){
  gtk_init(&argc, &argv);

  GTKWidget *pFenetre = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(pFenetre), "Les Labels");
  gtk_window_set_default_size(GTK_WINDOW(pFenetre), 320, 200);

  GTKWidget *pLabel = gtk_label_new("Back in Black!");
  gtk_container_add(GTK_CONTAINER(pFenetre), pLabel);
  gtk_widget_show_all(pFenetre);
  g_signal_connect(G_OBJECT(pFenetre), "destroy",
                   G_CALLBACK(detruire_fenetre), NULL);

  gtk_main();
  return EXIT_SUCCESS;
}//fin programme
```

Affichage (3)

```
$>gcc -o gtk gtk4.c `pkg-config --cflags --libs gtk+-2.0`
$>./gtk
```

Agenda

- Chapitre 1: Introduction aux Interfaces Graphiques
  - Programmation Evénementielle
  - GTK
  - Fenêtres
  - Labels
  - Boutons
    - ✓ Hiérarchie
    - ✓ Création
    - ✓ Signal
    - ✓ Exemple
  - Boxes
  - Menu
  - Tables

Hiérarchie

- La hiérarchie GTK pour les boutons est sensiblement identique à celle des fenêtres

```
GObject
  GtkObject
    GtkWidget
      GtkContainer
        GtkBin
          GtkButton
```


Création

- GTK considère 4 fonctions pour la création d'un bouton
  - `GtkWidget *gtk_button_new()`
    - ✓ bouton vide
    - ✓ personnalisation possible car un bouton est un container
  - `GtkWidget *gtk_button_new_with_label(const gchar *)`
    - ✓ bouton avec un label
  - `GtkWidget *gtk_button_new_with_mnemonic(const gchar *)`
    - ✓ bouton avec label et raccourci clavier
  - `GtkWidget *gtk_button_new_from_stock(const gchar *)`
    - ✓ bouton avec label, raccourci clavier et image (prédéfinie dans GTK)
    - ✓ `GtkStockItem` est une structure qui contient des informations prédéfinies

Signal

- Il existe un signal particulier qui est émis lorsque l'utilisateur clique sur le bouton
  - `clicked`
- En fonction de l'action à exécuter, il faut définir un callback particulier
- Exemple
  - bouton pour quitter

```
g_signal_connect(G_OBJECT(pQuitBtn), "clicked",
                 G_CALLBACK(gtk_main_quit), NULL);
```


Affichage

- Même principe que pour les labels
  - insertion du bouton dans la fenêtre
    - ✓ `gtk_container_add(GtkContainer *, GtkWidget *)`
  - affichage
    - ✓ `gtk_widget_show_all(GtkWidget *)`

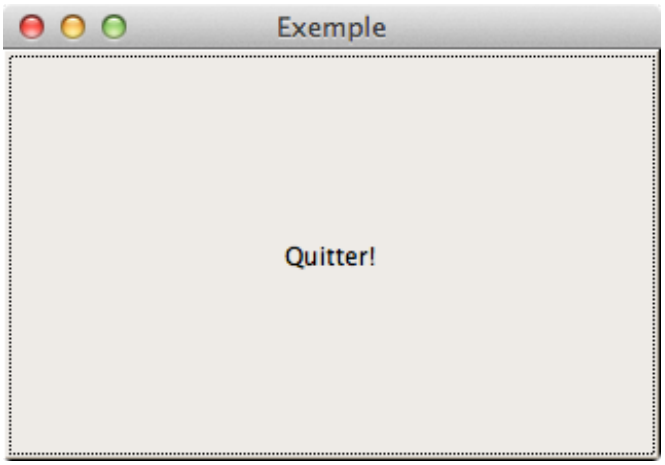
Exemple

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

void ajouter_btn(GtkWidget *pF){

  GTKWidget *pBoutonQuitter =
  gtk_button_new_with_label("Quitter!");
  gtk_window_set_title(GTK_WINDOW(pF), "Exemple");

  g_signal_connect(G_OBJECT(pBoutonQuitter), "clicked",
                   G_CALLBACK(gtk_main_quit), NULL);

  gtk_container_add(GTK_CONTAINER(pF), pBoutonQuitter);
}//fin ajouter_btn()
```


Exemple (2)

```
int main(int argc, char **argv){
  gtk_init(&argc, &argv);

  GTKWidget *pFenetre =
  gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_default_size(GTK_WINDOW(pFenetre), 320,
                              200);
  g_signal_connect(G_OBJECT(pFenetre), "destroy",
                   G_CALLBACK(gtk_main_quit), NULL);

  ajouter_btn(pFenetre);

  gtk_widget_show_all(pFenetre);

  gtk_main();

  return EXIT_SUCCESS;
}//fin programme
```

Exemple (3)



Agenda

- Chapitre 1: Introduction aux Interfaces Graphiques
  - Programmation Evénementielle
  - GTK
  - Fenêtres
  - Labels
  - Boutons
  - Boxes
    - ✓ Hiérarchie
    - ✓ Création
    - ✓ Insertion
    - ✓ Exemple
  - Menu
  - Tables

Hiérarchie

- Un `GtkContainer` ne peut contenir qu'un seul widget
  - Comment faire pour mettre plusieurs widgets dans un même widget?
- Solution
  - **Box**
- Hiérarchie

```
GObject
  GtkObject
    GtkWidget
      GtkContainer
        GtkBox
          GtkHBox
          GtkVBox
```


Création

- Création d'un `box` horizontal
  - `GtkWidget *gtk_hbox_new(gboolean, gint)`
    - ✓ TRUE pour diviser la box en *x* zones de taille égale
    - ✓ espace entre les widgets
- Création d'un `box` vertical
  - `GtkWidget *gtk_vbox_new(gboolean, gint)`
    - ✓ idem supra

Insertion

- Il n'y a pas de fonction spécifique, dans `GtkHBox` et `GtkVBox`, pour l'ajout de widget
  - il faut remonter à `GtkBox` pour trouver les fonctions
- Insertion de haut en bas

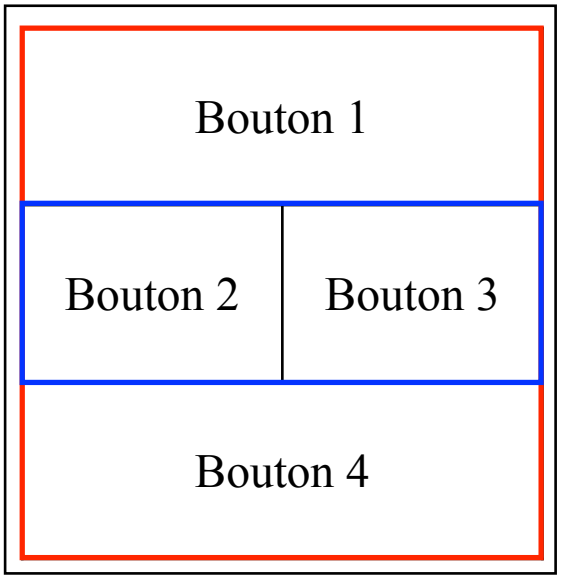
```
void gtk_box_pack_start(GtkBox *,        box à utiliser
                        GtkWidget *,     widget à insérer
                        gboolean,        dépend de la création
                        gboolean,        occupation
                        guint);          padding
```

- Insertion de bas en haut

```
void gtk_box_pack_end(GtkBox *,
                      GtkWidget *,
                      gboolean,
                      gboolean,
                      guint);
```


Exemple

GtkHBox



GtkVBox

Exemple (2)

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

GtkWidget *creer_fenetre(){
  GtkWidget *pF = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(pF), "Les Boxes");
  gtk_window_set_default_size(GTK_WINDOW(pF), 320, 200);

  g_signal_connect(G_OBJECT(pF), "destroy",
       G_CALLBACK(gtk_main_quit), NULL);

  return pF;
}//fin creer_fenetre()
```

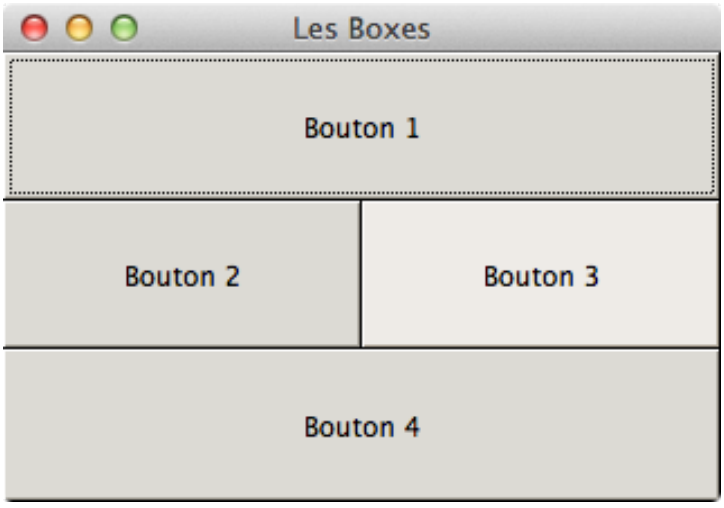

Exemple (3)

```
int main(int argc, char **argv){
  //voir slide précédent

  gtk_init(&argc, &argv);

  GTKWidget *pFenetre = creer_fenetre();

  GTKWidget *pVBox = gtk_vbox_new(TRUE, 0);

  GtkWidget *pBouton[4]
  pBouton[0] = gtk_button_new_with_label("Bouton 1");
  pBouton[1] = gtk_button_new_with_label("Bouton 2");
  pBouton[2] = gtk_button_new_with_label("Bouton 3");
  pBouton[3] = gtk_button_new_with_label("Bouton 4");

  gtk_box_pack_start(GTK_BOX(pVBox), pBouton[0], TRUE,
                     TRUE, 0);

  //à suivre
```

Exemple (4)

```
int main(int argc, char **argv){
  //voir slide précédent

  GTKWidget *pHBox = gtk_hbox_new(TRUE, 0);

  gtk_box_pack_start(GTK_BOX(pVBox), pHBox, TRUE, TRUE, 0);
  gtk_box_pack_start(GTK_BOX(pHBox), pBouton[1], TRUE,
                     TRUE, 0);
  gtk_box_pack_start(GTK_BOX(pHBox), pBouton[2], TRUE,
                     TRUE, 0);
  gtk_box_pack_start(GTK_BOX(pVBox), pBouton[3], TRUE,
                     TRUE, 0);

  gtk_container_add(GTK_CONTAINER(pFenetre), pVBox);
  gtk_widget_show_all(pFenetre);
  gtk_main();

  return EXIT_SUCCESS;
}//fin main()
```


Exemple (5)



Agenda

- Chapitre 1: Introduction aux Interfaces Graphiques
  - Programmation Evénementielle
  - GTK
  - Fenêtres
  - Labels
  - Boutons
  - Boxes
  - Menu
    - ✓ Hiérarchie
    - ✓ Menu Simple
    - ✓ Menu Complexe
    - ✓ Barre d'Outils
  - Tables

Hiérarchie

- La hiérarchie GTK pour les boutons est sensiblement identique à celle des fenêtres

```
GObject
  GtkObject
    GtkWidget
      GtkContainer
        GtkMenuShell
          GtkMenu
```

Menu Simple

- Il y a 3 widgets qui font partie d'un menu
  - **item**
    - ✓ ce que l'utilisateur sélectionne
    - ✓ e.g., "Ouvrir"
  - **menu**
    - ✓ container pour différents items
  - **menubar**
    - ✓ container pour chaque menu individuel

Menu Simple (2)

- Fonctions de créations
  - `GtkWidget *gtk_menu_bar_new();`
    - ✓ création d'une barre de menu
  - `GtkWidget *gtk_menu_new();`
    - ✓ création d'un menu
  - `GtkWidget *gtk_menu_item_new();`
    - ✓ création d'un item
  - `GtkWidget *gtk_menu_item_new_with_label(const char *);`
    - ✓ création d'un item avec un label
- Quand un item est créé, il faut l'attacher au menu
  - `gtk_menu_append(GtkMenuShell *, GtkWidget *);`
- Il est parfois nécessaire de passer par la notion de sous-menu
  - `gtk_menu_item_set_submenu(GtkMenuItem*, GtkWidget*);`
- Ne pas oublier d'attacher un signal à chaque item

Menu Simple (3)

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

GtkWidget *creer_fenetre(){
  GtkWidget *pF = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(pF), "Un Menu Simple");
  gtk_window_set_default_size(GTK_WINDOW(pF), 320, 200);

  g_signal_connect(G_OBJECT(pF), "destroy",
       G_CALLBACK(gtk_main_quit), NULL);

  return pF;
}//fin creer_fenetre()
```


Menu Simple (4)

```
GtkWidget *creer_menu(){

 //création des éléments
 GtkWidget *barre_menu = gtk_menu_bar_new();
 GtkWidget *menu_fichier = gtk_menu_new();
 GtkWidget *item_fichier =
 gtk_menu_item_new_with_label("Fichier");
 GtkWidget *item_quitter =
 gtk_menu_item_new_with_label("Quitter");

 //à suivre
}//fin creer_menu()
```

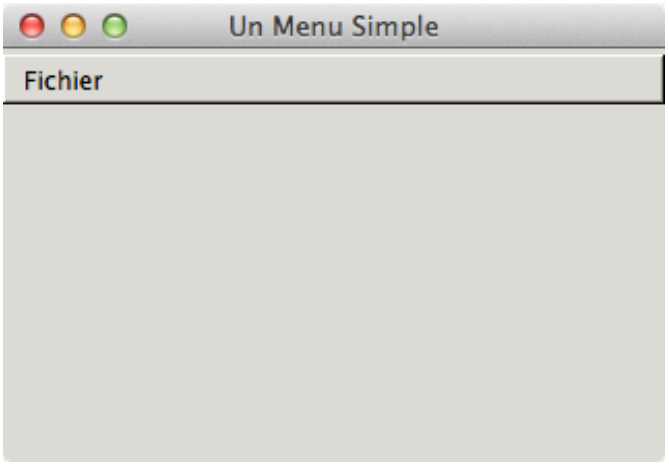
Menu Simple (5)

```
GtkWidget *creer_menu(){
  //voir slide précédent

  //attacher les items
  gtk_menu_item_set_submenu(GTK_MENU_ITEM(item_fichier),
                    menu_fichier);
  gtk_menu_shell_append(GTK_MENU_SHELL(menu_fichier),
                item_quitter);
  gtk_menu_shell_append(GTK_MENU_SHELL(barre_menu),
                item_fichier);

  //le signal
  g_signal_connect(G_OBJECT(item_quitter), "activate",
                  G_CALLBACK(gtk_main_quit), NULL);

  return barre_menu;
}//fin creer_menu()
```


Menu Simple (6)

```
int main(int argc, char **argv){

 gtk_init(&argc, &argv);

 GtkWidget *pFenetre = creer_fenetre();
 GtkWidget *pBarreMenu = creer_menu();

 GtkWidget *vBox = gtk_vbox_new(FALSE, 0);
 gtk_container_add(GTK_CONTAINER(pFenetre), vBox);
 gtk_box_pack_start(GTK_BOX(vBox), pBarreMenu, FALSE,
            FALSE, 3);

 gtk_widget_show_all(pFenetre);
 gtk_main();

 return EXIT_SUCCESS;
}//fin main()
```

Menu Simple (7)



Menu Complexe

- Il est possible d'avoir des raccourcis clavier pour un item
  - **accelerator**
- Hiérarchie

```
GObject
  GtkAccelGroup
```

Menu Complexe (2)

- Création d'un accelerator
  - `GtkAccelGroup *gtk_accel_group_new();`
- Association d'un accelerator à une fenêtre
  - `void gtk_window_add_accel_group(GtkWindow *,GtkAccelGroup *);`
- Liaison entre un accelerator et un signal

```
void gtk_widget_add_accelerator(GtkWidget *,
        const gchar *,
        GtkAccelGroup *,
        guint,
        GtkModifierType,
        GtkAccelFlags);
```

le signal
l'accelerator
touche du clavier
modificateur
drapeau

widget sur laquelle placer l'accelerator

Menu Complexe (3)

- On peut ajouter un raccourci pour accéder à un menu
  - `GtkWidget *gtk_menu_item_new_with_mnemonic(const gchar *);`
- On peut aussi ajouter une icône à un item
  - `GtkWidget * gtk_image_menu_item_new_from_stock(const gchar *, GtkAccelGroup *);`

Menu Complexe (4)

- Il est possible de séparer des groupes d'items dans un menu
  - **separator**
  - `GtkWidget *gtk_separator_menu_item_new();`

Menu Complexe (5)

```
#include <stdlib.h>
#include <gdk/gdkkeysyms.h>
#include <gtk/gtk.h>

GtkWidget *creer_fenetre(){
  GtkWidget *pF = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(pF), "Un Menu Complexe");
  gtk_window_set_default_size(GTK_WINDOW(pF), 320, 200);

  g_signal_connect(G_OBJECT(pF), "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  return pF;
}//fin creer_fenetre()
```

Menu Complexe (6)

```
GtkWidget *creer_menu(GtkWidget *pFenetre){

 GtkWidget *barre_menu = gtk_menu_bar_new();
 GtkWidget *menu_fichier = gtk_menu_new();

  GtkAccelGroup *accelerateur = gtk_accel_group_new();
  gtk_window_add_accel_group(GTK_WINDOW(pFenetre),
          accelerateur);


 //à suivre
}//fin creer_menu()
```


Menu Complexe (7)

```
GtkWidget *creer_menu(GtkWidget *pFenetre){
 //voir slide précédent
 //création des items avec images
 GtkWidget *item_fichier =
 gtk_menu_item_new_with_mnemonic("_Fichier");
 GtkWidget *item_nouveau =
 gtk_image_menu_item_new_from_stock(GTK_STOCK_NEW, NULL);
 GtkWidget *item_ouvrir =
 gtk_image_menu_item_new_from_stock(GTK_STOCK_OPEN, NULL);
 GtkWidget *item_separateur = gtk_separator_menu_item_new();
 GtkWidget *item_quitter =
 gtk_image_menu_item_new_from_stock(GTK_STOCK_QUIT,
         accelerateur);

 gtk_widget_add_accelerator(item_quitter, "activate",
                 accelerateur, GDK_q, GDK_CONTROL_MASK,
                 GTK_ACCEL_VISIBLE);
  //à suivre
}//fin creer_menu()
```

Menu Complexe (8)

```
GtkWidget *creer_menu(GtkWidget *pFenetre){
  //voir. slide précédent

 //attacher les items
 gtk_menu_item_set_submenu(GTK_MENU_ITEM(item_fichier),
   menu_fichier);
 gtk_menu_shell_append(GTK_MENU_SHELL(menu_fichier),
   item_nouveau);
 gtk_menu_shell_append(GTK_MENU_SHELL(menu_fichier),
   item_ouvrir);
 gtk_menu_shell_append(GTK_MENU_SHELL(menu_fichier),
   item_separateur);
 gtk_menu_shell_append(GTK_MENU_SHELL(menu_fichier),
   item_quitter);
 gtk_menu_shell_append(GTK_MENU_SHELL(barre_menu),
   item_fichier);

 //à suivre
}//fin creer_menu()
```

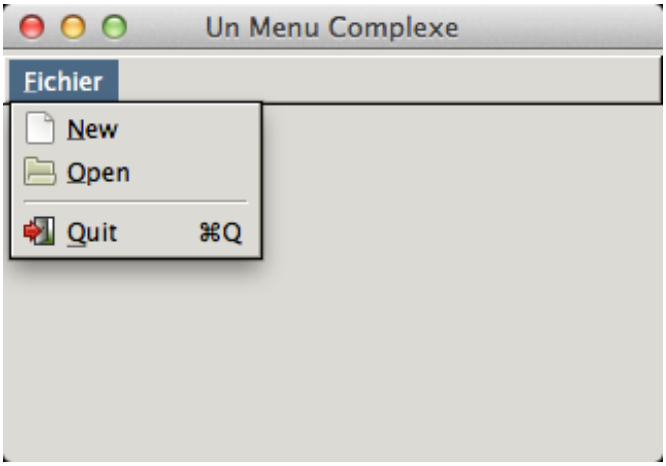

Menu Complexe (9)

```
GtkWidget *creer_menu(GtkWidget *pFenetre){
   //voir slide précédent

  //le signal
  g_signal_connect(G_OBJECT(item_quitter), "activate",
                   G_CALLBACK(gtk_main_quit), NULL);

  return barre_menu;
}//fin creer_menu()
```

Menu Complexe (10)

```
int main(int argc, char **argv){

  gtk_init(&argc, &argv);

  GtkWidget *pFenetre = creer_fenetre();
  GtkWidget *pBarreMenu = creer_menu(pFenetre);

  GtkWidget *vBox = gtk_vbox_new(FALSE, 0);
  gtk_container_add(GTK_CONTAINER(pFenetre), vBox);
  gtk_box_pack_start(GTK_BOX(vBox), pBarreMenu, FALSE,
             FALSE, 3);

  gtk_widget_show_all(pFenetre);
  gtk_main();

  return EXIT_SUCCESS;
}//fin main()
```


Menu Complexe (11)



Barre d'Outils

- La barre d'outils permet un accès rapide aux fonctionnalités les plus usitées
- Hiérarchie

```
GObject
  GtkObject
    GtkWidget
      GtkContainer
        GtkToolBar
```

- Création
  - `GtkWidget *gtk_toolbar_new();`
- Style de la barre d'outils
  - `void gtk_toolbar_set_style(GtkToolBar *, GtkToolBarStyle);`

Barre d'Outils (2)

- Ajout d'un item à la barre d'outils
  - `void gtk_toolbar_insert(GtkToolBar *, GtkToolItem *, gint);`

Barre d'Outils (3)

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

GtkWidget *creer_fenetre(){
  GtkWidget *pF = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(pF), "Un Menu Complexe");
  gtk_window_set_default_size(GTK_WINDOW(pF), 320, 200);

  g_signal_connect(G_OBJECT(pF), "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  return pF;
}//fin creer_fenetre()
```


Barre d'Outils (4)

```
GtkWidget *creer_barre_outils(){
 //création de la barre d'outils
 GtkWidget *barre_outils = gtk_toolbar_new();
 gtk_toolbar_set_style(GTK_TOOLBAR(barre_outils),
 GTK_TOOLBAR_ICONS);
 gtk_container_set_border_width(GTK_CONTAINER(barre_outils)
 , 2);
 //création items
  GtkToolItem *item_nouveau =
 gtk_tool_button_new_from_stock(GTK_STOCK_NEW);
  GtkToolItem *item_ouvrir =
 gtk_tool_button_new_from_stock(GTK_STOCK_OPEN);
  GtkToolItem *item_sauver =
 gtk_tool_button_new_from_stock(GTK_STOCK_SAVE);
  GtkToolItem *item_separateur =
 gtk_separator_tool_item_new();
  GtkToolItem *item_quitter =
 gtk_tool_button_new_from_stock(GTK_STOCK_QUIT); //à suivre
}//fin creer_menu()
```

Barre d'Outils (5)

```
GtkWidget *creer_barre_outils(){
  //voir slide précédent
   //ajout des items à la barre d'outils
  gtk_toolbar_insert(GTK_TOOLBAR(barre_outils), item_nouveau,
     -1);
  gtk_toolbar_insert(GTK_TOOLBAR(barre_outils), item_ouvrir, -1);
  gtk_toolbar_insert(GTK_TOOLBAR(barre_outils), item_sauver, -1);
  gtk_toolbar_insert(GTK_TOOLBAR(barre_outils), item_separateur,
     -1);
  gtk_toolbar_insert(GTK_TOOLBAR(barre_outils), item_quitter,
     -1);
  //à suivre
}//fin creer_menu()
```

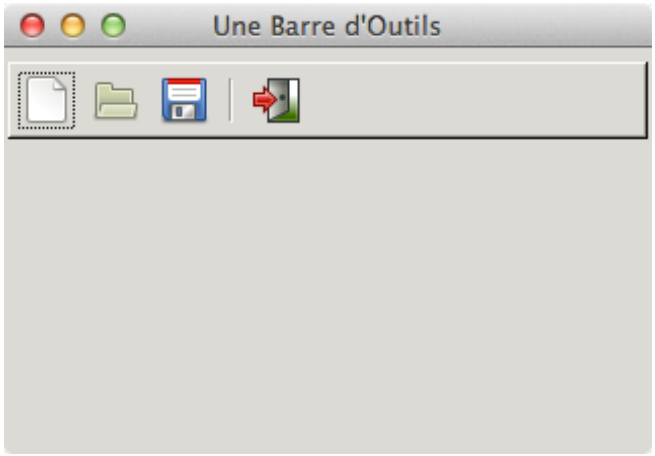

Barre d'Outils (6)

```
GtkWidget *creer_barre_outils(){
  //voir slide précédent

  //le signal
  g_signal_connect(G_OBJECT(item_quitter), "activate",
                   G_CALLBACK(gtk_main_quit), NULL);

  return barre_menu;
}//fin creer_menu()
```

Barre d'Outils (7)

```
int main(int argc, char **argv){

  gtk_init(&argc, &argv);

  GtkWidget *pFenetre = creer_fenetre();
  GtkWidget *pBarreOutils = creer_barre_outils();

  GtkWidget *vBox = gtk_vbox_new(FALSE, 0);
  gtk_container_add(GTK_CONTAINER(pFenetre), vBox);
  gtk_box_pack_start(GTK_BOX(vBox), pBarreOutils, FALSE,
             FALSE, 5);

  gtk_widget_show_all(pFenetre);
  gtk_main();

  return EXIT_SUCCESS;
}//fin main()
```


Barre d'Outils (8)



Agenda

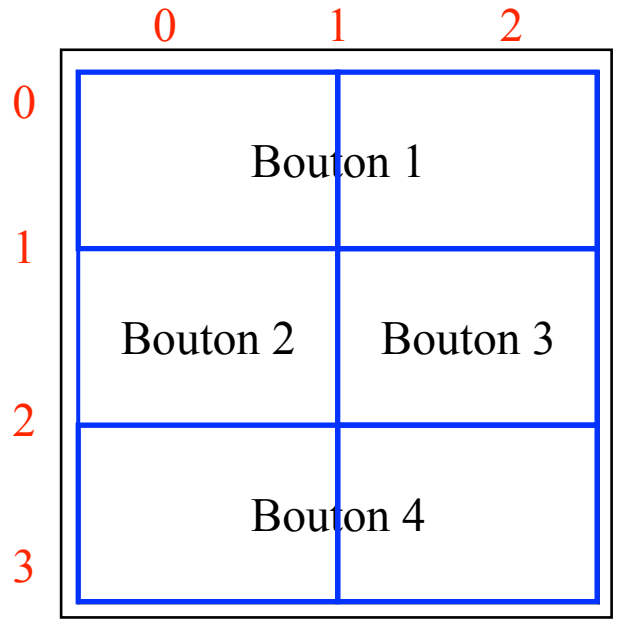
- Chapitre 1: Introduction aux Interfaces Graphiques
  - Programmation Evénementielle
  - GTK
  - Fenêtres
  - Labels
  - Boutons
  - Boxes
  - Menu
  - Tables
    - ✓ Hiérarchie
    - ✓ Création
    - ✓ Insertion
    - ✓ Modification
    - ✓ Exemple

Hiérarchie

- Placer plus de 4 boutons avec des `GtkBox` peut s'avérer fastidieux
- `GtkTable` utilise une grille invisible pour attacher les widgets
- Hiérarchie

```
GObject
  GtkObject
    GtkWidget
      GtkContainer
        GtkTable
```

Hierarchie (2)



Création

- La création se fait via

```
GtkWidget *gtk_table_new(guint,      # lignes
                         guint,      # colonnes
                         gboolean)   occupation
```

Insertion

- L'insertion d'une widget dans la `GtkTable` se fait via

```
void gtk_table_attach(GtkTable *,
                      GtkWidget *,
                      guint, guint,
                      guint, guint,
                      GtkAttachOptions,
                      GtkAttachOptions,
                      guint, guint)
```

la table
la widget
colonnes
lignes
occupation
padding

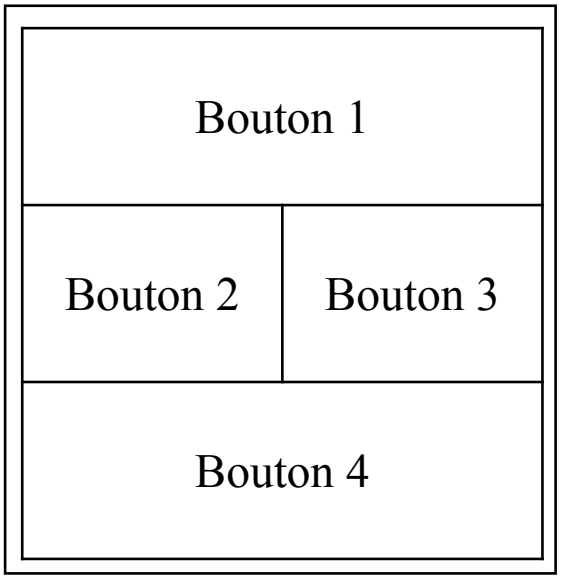
Insertion (2)

- Occupation?
  - `GTK_EXPAND`
    - ✓ la grille s'étire pour remplir l'espace disponible
  - `GTK_FILL`
    - ✓ le widget s'étire pour remplir l'espace disponible
- Combinaisons possibles
  - `GTK_EXPAND | GTK_FILL`

Modification

- Il est possible de modifier la taille d'une grille après sa création
  - `void gtk_table_resize(GtkTable *, guint, guint)`
- On peut aussi changer l'espace d'une ligne ou colonne spécifique
  - `void gtk_table_row_spacing(GtkTable *, guint, guint)`
  - `void gtk_table_col_spacing(GtkTable *, guint, guint)`

Exemple



Exemple (2)

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

GtkWidget *creer_fenetre(){
  GtkWidget *pF = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(pF), "Les Tables");
  gtk_window_set_default_size(GTK_WINDOW(pF), 320, 200);

  g_signal_connect(G_OBJECT(pF), "destroy",
  G_CALLBACK(gtk_main_quit), NULL);

  return pF;
}//fin creer_fenetre()
```


Exemple (3)

```
int main(int argc, char **argv){
  //voir slide précédent

  gtk_init(&argc, &argv);

  GtkWidget *pFenetre = creer_fenetre();

  GtkWidget *pTable = gtk_table_new(3, 2, TRUE);

  GtkWidget *pBouton[4];
  pBouton[0] = gtk_button_new_with_label("Bouton 1");
  pBouton[1] = gtk_button_new_with_label("Bouton 2");
  pBouton[2] = gtk_button_new_with_label("Bouton 3");
  pBouton[3] = gtk_button_new_with_label("Bouton 4");

  //à suivre
```

Exemple (4)

```
int main(int argc, char **argv){
  //voir slide précédent

  gtk_table_attach(GTK_TABLE(pTable), pBouton[0], 0, 2, 0,
                   1, GTK_EXPAND, GTK_EXPAND, 0, 0);
  gtk_table_attach(GTK_TABLE(pTable), pBouton[1], 0, 1, 1,
                   2, GTK_EXPAND, GTK_EXPAND, 0, 0);
  gtk_table_attach(GTK_TABLE(pTable), pBouton[2], 1, 2, 1,
                   2, GTK_EXPAND, GTK_EXPAND, 0, 0);
  gtk_table_attach(GTK_TABLE(pTable), pBouton[3], 0, 2, 2,
                   3, GTK_EXPAND, GTK_EXPAND, 0, 0);

  gtk_container_add(GTK_CONTAINER(pFenetre),
         GTK_WIDGET(pTable));
  gtk_widget_show_all(pFenetre);
  gtk_main();

  return EXIT_SUCCESS;
}//fin main()
```


Exemple (5)