Programmation d'interfaces

Cours 1 - Introduction à *GTK* et initiation à la *GLib*

H. Djerroud

LIASD - Université Paris 8

Automne 2020

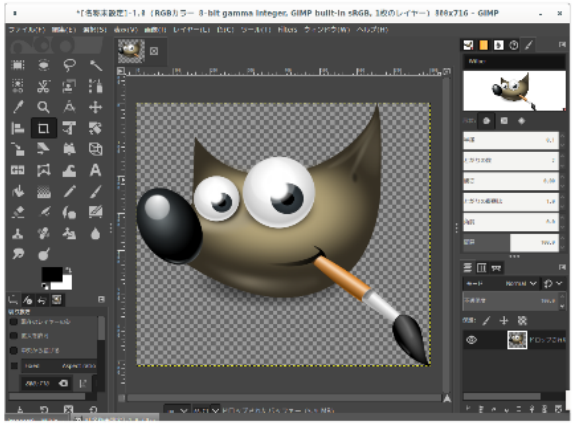
Plan de cours

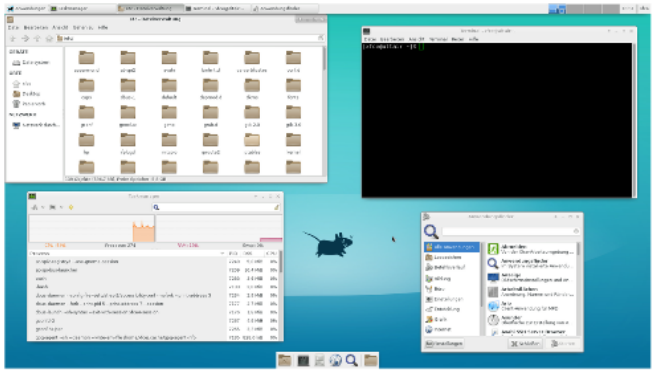
- Introduction à *GTK* et les bibliothèques concurrentes
- Introduction à la *GLib* et la différence avec *glibc*
- Compiler un programme *GLib*
- Les types de bases et les macros fournis par la *GLib*
- Afficher des message de Log et gestion des erreurs
- Gestion de la mémoire
- Gestion des fichiers
- Les timers
- La boucle principale
- Les structures de données fournies par *GLib* (chaînes de caractères, listes chaînées, tableaux, arbres binaires, etc.

Introduction à *GTK*

- *GTK* pour **G**IMP **T**ool**k**it, distribuée sous la licence **LGPL**
- Un ensemble de bibliothèques permettant de réaliser des interfaces graphiques multiplateforme
- Initialement développée pour les besoins du logiciel de traitement d'images *GIMP*
- Utilisée dans les projets : *Xfce*, *GNOME* ...
- Disponible sur plusieurs langages : C, C++ (*gtkmm*), php ...
- *GTK* se base sur trois bibliothèques :
  - **GLib** : bibliothèque pour les traitements basiques, structures de données, gestionde la mémoire ...
  - **Pango** : bibliothèque pour le rendu de texte
  - **Cairo** : bibliothèque pour dessin 2D
  - **ATK** : interfaces d'accessibilité (clavier virtuel, loupe, ...)

Quelques exemples





Les bibliothèques concurentes de *GTK*

- Tk (**T**ool**k**it), Tcl/Tk
- Qt prononcé *cute*
- Swing
- wxWidgets
- FLTK (**F**ast **L**ight **T**ool**k**it)
- ...

Avantages d'utiliser *GTK* :

- Licence LGPL, existe sur plusieurs plates-formes et utilisable avec plusieurs langages de programmation
- GNU/Linux a tendance à être plus orienté GTK
- Une boîte à outils standard pour GNOME

Introduction à la *GLib* et la différence avec *glibc*

- Écrite en C spécialement pour la *GTK* : à l'origine elle est destinée à faciliter le portage du code *GTK* sur différentes plates-formes (taille des types des variables, boutisme, structures de données . . .)
- L'objectif est de séparer les fonctions non-graphiques de GTK dans une bibliothèque indépendante (depuis la version 2) : à l'origine ces fonctions sont incluses dans *GTK*
- Ne pas confondre avec *glibc ou libc*, l'objectif n'est pas de récrire la *libc*, mais s'utilise en complément
- Peut être utilisée indépendamment de la librairie *GTK*

Contenu de la *GLib*

- Toutes les fonctions, définition et macros sont préfixées par : `g*`, `G*`, `g_*` ou `G_*`
- Offre une interface Orientée Objet pour le langage C (*GObject*) *qui n'existe pas en langage C*
- Implémentation portable des types fondamentaux. : e.g. `gint` équivalent à *int*, mais garde la même représentation quel que soit la plate-forme . . .
- Structures de donnée : listes chaînées, files, arbres, tables de hachage.
- Fonctions sécurisées pour les chaînes et tableaux
- Définition d'un type `string` pour C `g_string`

Contenu de la *GLib* (suite)

- Fonction pour pour l'interaction avec l'utilisateur : réimplémentation de `getopt()`.
- Allocation mémoire avec gestion des erreurs
- Fonctions pour analyse de fichiers structurés courants (fichiers de conf, XML ...)
- ...

Premier contact avec GLib

- Installer GLib sur votre machine :
  `apt-get install libgtk2.0-dev`
- Connaître la version de GLib installée sur votre machine :
  `pkg-config --modversion glib-2.0`
- Documentation officielle :
  `https://developer.gnome.org/glib/`
- Inclure : `glib.h` dans votre programme

Premier programme

```
/* exemple1.c */
#include <glib.h>
int main (int argc, char* agrv[]){
  g_print ("Hello World \n");
  return 0;
}
```

Compiler une programme GLib

- Obtenir les fichiers essentiels pour la compilation :
  `pkg-config --cflags --libs glib-2.0`
- Pour compiler :
  `gcc ‘pkg-config --cflags --libs glib-2.0‘ example1.c -o exemple1`

Les types de base

Type	Description
gboolean	TRUE ou FALSE
gchar, guchar	≡ à char et unsigned char en C standard
gdouble	≡ à double en C, [G_MINDOUBLE et G_MAXDOUBLE]
gfloat	≡ à float en C, [G_MINFLOAT et G_MAXFLOAT]
gint, guint	≡ à int et unsigned int en C, [G_MININT et G_MAXINT]
gint8	int sur 8 bits signé et non signé
gint16	int sur 16 bits signé et non signé
gint32, guint32	int sur 32 bits signé et non signé
gint64, guint64	int sur 32 bits signé et non signé
gpointer	Pointeur générique non typé défini comme void *
gsize, gssize	Entiers 32 bits non signés et signés utilisés par de les structures de données pour représenter les tailles.

Avantages à utiliser ces types de base

L'un des principaux avantages de l'utilisation de *GLib* est qu'elle fournit une interface multiplateforme. Les types présentés ici fournissent un ensemble de **types de données de base** qui sont :

- Portables sur d'autres plates-formes
- Portables sur d'autres langages de programmation utilisant *GLib*/*GTK*.

Les macros standards GLib

Macro	**Description**
ABS(x)	Retourne la valeur absolut
CLAMP(a,low,high)	Retourne TRUE si a est entre low et high
G_DIR_SEPARATOR	Retourne le séparateur de répertoire utilisé (OS)
GINT_TO_POINTER(i)	Converti un entier en pointeur
GPOINTER_TO_INT(p)	Converti un pointeur en un entier
MIN(a,b), MAX(a,b)	Retourne le min/max entre a et b
TRUE, FALSE	FALSE est définie à 0 et TRUE est l'inverse
...	...

Les constantes mathématiques

- Les macros utilisent généralement une précision de 50 décimales

Constante	**Description**
G_E	La base du logarithme naturel (2.718282...)
G_LN2	Le logarithme naturel de 2
G_LN10	Le logarithme naturel de 10
G_PI	π
G_PI_2	$\pi/2$
G_PI_4	$\pi/4$
G_LOG_2_BASE_10	Le logarithme de 2 en base 10
G_SQRT2	La racine carrée de 2

Les macros GLib

Valeur	Description
GLIB_MAJOR_VERSION	Version Majeurde GLib
GLIB_MINOR_VERSION	Version Mineur GLib
GLIB_MICRO_VERSION	Version Mirci GLib
CHECK_VERSION (major, minor, micro)	Return True si la version du système est égale ou supérieur à la valeur indiquée en paramètre

Afficher un message à l'écran

- Pour afficher des message à l'écran on utilise la fonction : `void g_print (const gchar *format, ...);`
- Cette fonction ne doit pas être utilisé pour les message de débogage, pour cela on utilise plutôt `g_log()`, `g_log_structured()`, `g_message()`, `g_warning()` ou `g_error()`
- Il existe d'autres fonction d'affichage spécifiques e.g. `g_set_print_handler()`, `g_printerr()`, `...`, se référer à la documentation officielle pour avoir plus d'informations.

Les logs

- Pour afficher des logs il faut utiliser la fonction suivante :

```
void g_log (const gchar *log_domain,
            GLogLevelFlags log_level,
            const gchar *message,
            ...);
```

- Le `log_domain` est une chaîne de caractère utilisée pour aider l'utilisateur à différencier les messages générés par votre application de ceux générés par d'autres bibliothèques.
- Permet de spécifier le type de message (niveaux de log).

Les logs : les niveaux de logs

log_level	Description
G_LOG_FLAG_RECURSION	Utilisé pour les messages récursifs
G_LOG_FLAG_FATAL	Erreurs entraîneront la fermeture de l'application et la remise à 0 du core lors de l'appel.
G_LOG_LEVEL_ERROR	Un type d'erreur qui est toujours fatal.
G_LOG_LEVEL_CRITICAL	Une erreur non fatale qui est plus importante qu'un WARNING mais qui ne ferme pas l'application.
G_LOG_LEVEL_WARNING	Avertissement qui n'empêche pas l'app de continuer.
G_LOG_LEVEL_MESSAGE	Messages normaux qui ne sont pas critiques.
G_LOG_LEVEL_INFO	Tout autre type de message non couvert par les autres niveaux, comme les informations générales.
G_LOG_LEVEL_DEBUG	Un message général utilisé à des fins de débogage.
G_LOG_LEVEL_MASK	G_LOG_FLAG_RECURSION | G_LOG_FLAG_FATAL

Autres fonctions de log

- On peut aussi utiliser cinq fonctions qui permettent d'afficher un *message* sans fournir un `log_domain`.

```
void g_message (...); /* G_LOG_LEVEL_MESSAGE */
void g_warning (...); /* G_LOG_LEVEL_WARNING */
void g_critical (...); /* G_LOG_LEVEL_CRITICAL */
void g_error (...);     /* G_LOG_LEVEL_ERROR */
void g_debug (...);     /* G_LOG_LEVEL_DEBUG */
```

Gestion de la mémoire

- GLib fourni un ensemble de fonctions permettant la manipulation de la mémoire : allocation, libération, redimensionnement, etc.
- Des fonctions plus simple à utiliser et plus sûr
- Possibilité d'allouer/libirer plus efficacement des blocs de mémoire

Les allocators

- *GLib* fourni des fonctions équivalent aux fonctions *malloc* de libc on trouve par exemple :

```
struct_type* g_new (struct_type, number_of_structs)
```

- Utilisée pour allouer `number_of_structs` des structure de type `struct_type`, la fonction retourne un pointeur. Pour initialiser les donner à 0 il existe une autre fonction avec la même signature : `g_new0(...)`

```
gpointer g_malloc (gulong number_of_bytes)
```

- Une version portable de `malloc`, il suffi d'indiquer le nombre d'octet a allouer, la fonction retourne un pointeur. in existe aussi une version : `g_malloc0`

Les allocators (suite)

`gpointer g_try_malloc (gulong number_of_bytes)`

- Une version sécurisée de `malloc`, en cas d'échec d'allocation méméorie la fonction return `NULL`

`void g_free (gpointer memory)`

- Libère la mémoire à la manière de `free()`, si un pointeur `NULL` est passé en paramètre alors la fonction est simplement ignorée

`g_memmove (dest,src,len)`

- Copie un bloc de mémoire de len octets de long, de src à dest. Les zones source et destination peuvent se chevaucher

Allouer des blocs de mémoire

- *GLib* fourni aussi des fonction d'allocation efficaces (réduit la fragmentation) de blocs de mémoire

```
gpointer g_slice_alloc (gsize block_size)
```

- Permet d'allouer efficacement un certain nombre de `block_size` blocs mémoires
- Les blocs alloués avec cette fonction ne doivent pas être redimensionnés

```
void g_slice_free1 (gsize block_size, gpointer mem_block)
```

- Permet de libérer la mémoire allouée avec `g_slice_alloc()`

Exemple g_slice_alloc

```
gchar *mem[10000];
gint i;
for (i = 0; i < 10000; i++){
    mem[i] = g_slice_alloc (50);
    for (j = 0; j < 50; j++)
        mem[i][j] = i * j;
}
for (i = 0; i < 10000; i++)
    g_slice_free1 (50, mem[i]);
```

Compter le temps

- Glib fournit un ensemble de fonctions permettant de gérer un Timer :

```
GTimer * g_timer_new ()
```

- Créé un nouveau timer, le démarre dans la foulée et retourne son adresse

```
void g_timer_start (GTimer *timer)
```

- Démarre le timer à l'adresse **timer*

```
void g_timer_stop (GTimer *timer)
```

- Arrête le timer

Compter le temps (suite)

```
gdouble g_timer_elapsed (GTimer *timer, gulong
*microseconds)
```

- Si le timer a été démarrée mais pas arrêtée, le temps écoulé sera calculé en fonction de l'heure de début. Si `g_timer_continue()` a été utilisé pour redémarrer le timer, les deux temps seront additionnés pour calculer le temps total écoulé. La valeur de retour de `g_timer_elapsed()` est le nombre de secondes qui se sont écoulées avec tout temps fractionnaire. Le second paramètre qui renvoie le nombre de microsecondes écoulées, ce qui est généralement inutile puisque nous avons déjà récupérer le nombre de secondes sous forme de valeur à virgule flottante.

Compter le temps (suite)

`void g_timer_reset (GTimer *timer)`

- Redémarre le timer si il a été arrêté

`void g_timer_destroy (GTimer *timer)`

- Libère la mémoire utilisée par le timer

`gboolean g_timer_is_active (GTimer *timer)`

- Retourne `TRUE` qi le time est active, `FALSE` sinon

`void g_timer_continue (GTimer *timer)`

- Créer un nouveau timer, le démarre dans la foulée et retourne le timer créé

Exemple d'utilisation d'un Timer

```
#include <glib.h>
int main (int argc, char* argv[]){
  // génération d'un nombre aléatoire
  GRand* grand = g_rand_new();
  gint irand = (g_rand_int (grand) % 10) *  1000000;
  g_rand_free(grand);
  GTimer *timer = g_timer_new (); // création d'un timer
  g_usleep (irand);               // faire dodo
  g_timer_stop(timer);            // on arrte le timer
  // On compte le temps écoulé
  gdouble time_sleep = time_sleep = g_timer_elapsed(timer,NULL);
  g_print(" time_sleep  %f \n", time_sleep );
  g_timer_destroy(timer);        // On détruit le timer
  return 0;
}
```

XXX

- XX

XXX

Constante	Description
G_FILE_TEST_IS_REGULAR	Ce n'est pas un lien symbolique ou un répertoire, ce qui signifie qu'il s'agit d'un fichier normal
G_FILE_TEST_IS_SYMLINK	Lien symbolique.
G_FILE_TEST_IS_DIR	Répertoire
G_FILE_TEST_IS_EXECUTABLE	Fichier éxecutable
G_FILE_TEST_EXISTS	Un certain type d'objet existe à l'emplacemen (Fichier, lien symbolique, répéroite ou fichier normal)

Boucle principale

- Le but de la boucle principale est de se mettre en pause jusqu'à ce qu'un évènement se soit produit
- *GLib* utilise l'appel système `pool()`, ainsi les évènements et les signaux sont associé à un descripteur de fichier
- La boucle principale *GLib* est implémentée sous la forme d'un certain nombre de structures, qui permettent à plusieurs instances d'être exécutées simultanément. `GMainContext` est utilisé pour représenter un certain nombre de sources d'événements. Chaque thread a son propre contexte, qui peut être récupéré avec `g_main_context_get()`. Il est possible récupérer le contexte par défaut avec `g_main_context_get_default()`

Les Timeouts

- Les timeouts sont des méthodes appelées à un certain intervalle de temps jusqu'à ce que FALSE soit retourné
- Pour les ajouter à la boucle principale : `g_timeout_add_full()` ou `g_timeout_add()`

Les Timeouts : exemple

```
#include <glib.h>
gboolean timeout_callback(gpointer data) {
    static int i = 0;
    i++;
    g_print("timeout_callback called %d times\n", i);
    if (10 == i) {
        g_main_loop_quit( (GMainLoop*)data );
        return FALSE;
    }
    return TRUE;
}
int main(int argc, char* argv){
  GMainLoop * mloop = g_main_loop_new ( NULL , FALSE );
  g_timeout_add (100 , timeout_callback , mloop);
  g_main_loop_run (mloop);
  g_main_loop_unref(mloop);
  return 0;
}
```

Le type String

- *GLib* dispose d'un type String qui facilite grandement le travail du développeur C, il est définie dans une structure :

```
typedef struct {
    gchar *str;
    gsize len;
    gsize allocated_len;
} GString;
```

In ne faut faire de référence permanente au membre str d'une GString. Il peut être déplacé vers un emplacement mémoire différent au fur et à mesure que du texte est ajouté ou inséré ou supprimé de la chaîne en raison d'un changement dans la longueur allouée de la chaîne !

Créer une String

Il existe trois façon de créer une `GString` :

`GString* g_string_new(const gchar *initial_str)`

- Crée un nouveau GString, initialisé avec la chaîne donnée

`GString* g_string_new_len(const gchar *initial_str,gssize length)`

- Initialise la chaîne GString avec les caractères de longueur de la chaîne initiale ou la chaîne entière si la longueur est -1.

`GString* g_string_sized_new (gsize default_size)`

- Crée une nouvelle GString, avec suffisamment d'espace pour `default_size` octets. Ceci est utile si vous devez ajouter beaucoup de texte à la chaîne et vous ne souhaitez pas qu'il soit réalloué trop souvent.

Afficher une String

```
void g_string_printf (GString *string,
                      const gchar *format,
                      ...);
```

- La fonction *g_string_printf(...)* est équivalente à la fonction `sprintf()` de la *libc*

Ajouter du texte à une String

GString g_string_append (GString *string,const gchar *val) ;*

- xxx

GString g_string_append_len ((GString *string, const gchar *str, gssize len)*

- xxx

GString g_string_append_c (GString *string,gchar c) ;*

- xxx

GString g_string_append_unichar (GString *string,gunichar wc) ;*

- xxx

XXX

- XX

XXX

- XX