GLib

Hans Ole Hatzel

5. Juli 2013

Inhaltsverzeichnis

1 Einleitung 2
1.1 Geschichte und Einsatzzwecke 2
1.2 Benutzung 2

2 Funktionen 2
2.1 Funktionsübersicht 2
2.2 Datentypen und Datenstrukturen 3
2.3 LinkedLists 3
2.4 Macros 5
2.5 Eventloop 5
2.6 Threads 6
2.7 Weitere Funktionen 8
2.7.1 GObject 8
2.7.2 Utilities 8

3 Vorteile und Nachteile 9

4 Quellen 9

1 Einleitung

1.1 Geschichte und Einsatzzwecke

Die GLib (Gnome Library) wird vom Gnome Projekt entwickelt. Das Gnome Projekt ist vor allem für die Gnome Desktopumgebung bekannt. Die Glib entstand bei der Entwicklung des GUI Frameworks GTK+.
Zuerst war die GLib Teil von GTK, doch man gliederte die GLib aus, sodass heute die Funktionen der GLib auch in nicht GUI-Software, ohne GTK, nutzbar sind. [2]
Die Gnome Library enthält deswegen viele Funktionen zur GUI Entwicklung, ein weiterer Fokus liegt auf der Crossplatform Entwicklung, also der Unterstützung verschiedener Hard- und Software Plattformen.
Wichtig ist es, den Unterschied klarzustellen, die GLib ist *keine* Implementation der Standart C Library (wie etwa die glibc) sondern eine unabhängige Library mit eigenen Ansprüchen.

1.2 Benutzung

Um die GLib zu verwenden muss sie wie jede Library über ein Include Statement eingebunden werden.

```
#include <glib.h>
```

Außerdem muss dem Compiler mittels pkg-config mitgeteilt werden, wo er die entsprechenden Library Datein (in diesem Fall der Glib) zu finden hat.
Das kann über den folgenden Aufruf aus der Linux Konsole geschehen:

```
$ gcc example.c $(pkg-config --cflags --libs glib-2.0)
```

Dem Compiler, hier GCC, wird neben der Eingabedatei auch per pkg-config der Speicherort übergeben, an dem sich die GLib befindet. Dieser komplizierte Aufruf wird natürlich in der Regel durch ein Makefile getätigt und mit anderen Buildoptionen kombiniert.

2 Funktionen

2.1 Funktionsübersicht

Da die GLib ihre Ursprünge in der GUI Programmierung findet sind auch viele ihrer Funktionen die vor allem bei der GUI Programmierung gedacht, allerdings finden sich auch viele Funktionen die im allgemeinen nützlich sind und

die häufig verwendet werden. Ähnlich wie viele Funktionen die in Java bereits Standartmäßig vefügbar sind, wie etwa Linked Lists. Diese müssen dann nicht immer wieder selbst implementiert werden, was natürlich die Komplexität des Codes verringert und eventuell bessere Performance bietet, was besonders bei komplexeren Anwendungen wie dem sortieren von Listen von Vorteil sein kann.
Im folgenden werde ich einige der Funktionen die ich für besonders interessant und wichtig halte detailliert behandeln.

2.2 Datentypen und Datenstrukturen

Die GLib bietet für viele eigenen primitive Datentypen diese unterscheiden sich kaum von den Standard C Implementationen, werden allerdings als Eingabe- und Ausgabewerte für Funktionen der GLib verwendet. Man unterscheidet sie anhand des g-Prefixes von Standard C Datentypen. Ein paar Beispiele:

- gbool (ohne stdbool.h benutzbar)
- gint
- gint8 (Einfacher Zugriff auf eine garantiert 8 Bit breite Integer)

Datenstrukturen sind wie der Name schon vermuten lässt Strukturen aus mehreren Datentypen, hier bietet die Standart C Libarary kaum etwas, umso wichtiger sind die Datenstrukturen der GLib die ich zum Teil im weiteren näher erklären werde. Hier schonmal einige Beispiele aus der GLib:

- LinkedLists
- HashTables
- Bäume
- Wachsende Arrays
- Strings

2.3 LinkedLists

Linked Lists sind eine Darstellungsform von Listen als Datenstruktur, dabei wird jedem Eintrag aus einer Liste sein Wert, sowie ein Vorgänger und ein Nachfolger zugewiesen. Dies hat den Vorteil das man von jedem Element einfach seinen Nachfolger und Vorgänger erreichen kann, wie es auch bei

Abbildung 1: Doppelt verlinkte Liste

einem Array ist. Zugleich kann man jedoch solche Listen theoretisch beliebig erweitern. Nicht nur an den Enden an denen noch freie Speicherbereich ist, sondern an beliebiger Stelle, es lassen sich also ganz einfach neue Elemente an allen Stellen in der Liste einfügen. Im Vergleich zum Array sind dafür Zugriffe auf ein bestimmtes Element viel langsamer, man kann nicht einfach auf z.B das 53. Element zugreifen, sondern muss von einer Seite über alle Elemente die vor bzw. nach dem 53. stehen iterieren.

Eine solche Datenstruktur lässt sich ohne weiteres in C selber implementieren, allerdings stellt die GLib noch weitere Funktionen zur Verfügung die den Umgang mit Linked Lists vereinfacht. Intern wird ein Element der Liste als Struct dargestellt:

```
struct GList {
  gpointer data;
  GList *next;
  GList *prev;
};
```

Die Pointer zeigen jeweils auf die Adresse an der ein weiteres Element der Liste liegt, entsprechen also den Pfeilen in der Beispielabbildung. Ein weiterer Pointer verweist auf die Daten die ein Element enthält.

Die GLib stellt nun Funktionen zur Verfügung um mit den Elementen abstrakt zu arbeiten, der Entwickler nutzt die Liste nur noch als ganzes um Beispielsweise Elemente anzuhängen.

```
1  /* Die Liste wird auf den Wert Null initialisier.*/
2  GList *liste = Null;
3
4  /* Strings werden an Anfang und Ende der Liste eingefuegt.*/
5  liste = g_list_append(liste, "bli");
6  liste = g_list_append(liste, "bla");
7  liste = g_list_prepend(liste, "blup");
8
9  gchar *x = g_list_nth_data(liste, 0);
10 printf("value: %s", x); // Ausgabe: "blup"
```

Noch interessanter wird die Verwendung der Linked List Implementation aus der GLib wenn man auf komplexere Funktionen angewiesen ist.

```
//Eine Funktion auf jedes Element der Liste ausfuehren.
g_list_foreach(GList *list, GFunc func, gpointer user_data)

//Zwei Listen aneinander haengen.
g_list_concat(GList *list1, GList *list2)

//Einen Wert in einer Liste suchen.
g_list_find(GList *list, gconstpointer data)
```

2.4 Macros

Macros bieten eine Vielzahl an Funktionen die durch den Präprozessor verarbeitet werden. In der GLib gibt es dazu die Üblichen Macros die etwa die Version der Library oder das Betriebssystem ausgeben. Außerdem gibt es zum Beispiel noch das Macros zur Endian Konvertierung. Damit können also Bit Reihenfolgen auf andere Systeme angepasst werden.

2.5 Eventloop

Eventloops können immer dann eingesetzt werden wenn Programme zur Laufzeit Usereingaben erhalten. Denn man kann sich in diesem Fall nicht einfach auf gegebene Startparameter verlassen, sondern muss jederzeit auf Input reagieren können. Die Eventloop funktioniert dann so das in einer Schleife immer wieder Eingabe verarbeitet und entsprechende Operationen ausgeführt.
Anwendugsfälle sind also z.B. GUI Anwendungen oder Spiele. Da die GLib aus der GUI Programmierung kommt hat sie eine Eventloop.
Als Pseudocode sieht das dann so aus:

```
while(running){ //Sobald das Programm gestartet ist
    event = getEvent() //werden Events gegettet ...
    process(event) // ... und verarbeitet
}
```

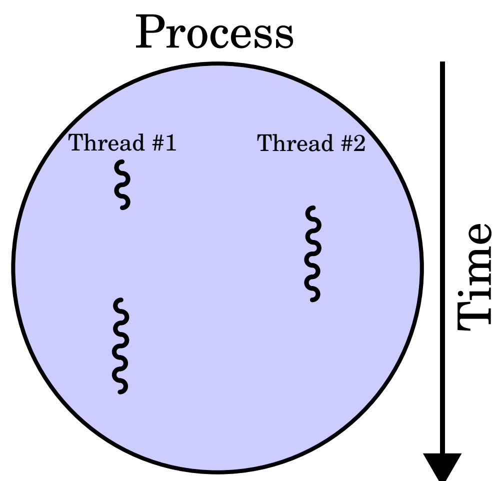
Eventloops in GLib werden immer zunächst erstellt, dann mit Events versehen und im letzten Schritt gestartet woraufhin sie Funktionen auslösen sobald Events eintreten. Im folgenden einfachen Beispiel wird dabei einfach nur ein Timer in die Eventloop gesetzt, bei erreichen der gegebenen Zeit wird dann „callback" ausgeführt. [3]

```
gboolean callback(gpointer data){
    g_print("timeout ");
    return TRUE;
}

int main(){
    GMainLoop *schleife;
    schleife = g_main_loop_new(NULL, FALSE);
    //Schleife erstellen

    g_timeout_add(1000, callback, NULL); //Event hinzufuegen
    g_main_loop_run(schleife); //Schleife startem
} //Ausgabe: timeout timeout timeout ...
```

2.6 Threads

Threads sind eine Methode um parallele Vorgänge auf linear ablaufenden Systemen auszuführen. So kann es in einem Prozess mehrere Threads geben, die die parallel ausgeführt werden. Dies gescheit auf Prozessoren mit mehreren Kernen nicht nur simuliert (durch immer wieder kurzes unterbrechen des einen Ablaufs) sondern auch tatsächlich parallel.

Abbildung 2: Threads eines Prozesses [4]



Ein Vorteil gegenüber getrennten Prozessen ist das die Threads in einem Prozess sich den gleichen Speicher teilen und somit keine Interprozess

Kommunikation von Nöten ist (was eher kompliziert ist). Da zwei unterschiedliche Abläufe auf den gleichen Speicher zugreifen ist danz ganze aber auch anfällig für Bugs. Es können natürlich unerwartete Zustände des Speichers auftreten wenn ein anderer Thread gerade Werte verändert hat, allerdings kann auch durch gleichzeitigen Zugriff beider Threads ein undefinierter Zustand (Racecondition) im Speicher auftreten. In GLib wird deshalb Speicher standardmäßig gelockt, also für andere Threads gesperrt wenn darauf gerade durch einen Thread des Prozesses zugegriffen wird. Allerdings findet das zwar bei normalen Variablen, nicht aber bei allen Datenstrukturen statt. Desswegen muss man beispielsweise bei der Verwendung von Hashtabels trotzdem darauf acht geben Threadsafe zu programmieren.
Warum sollte man nun die Thread Funktionen der GLib verwenden? Zum einen natürlich weil vieles schon standartmässig Threadsafe ist, zum anderen aber auch weil die GLib Threads auf unterschiedlichen Betriebssystem bzw. Plattformen gleich funktionieren, also die unterschiedlichen Schnittstellen der Betriebssysteme abstrahiert werden. Somit ist der gleiche (Multithread-)Code im Idealfall sowhol auf Unix als auch Windows lauffähig.

```
1  int k = 20;
2
3  gpointer writeToMem(gpointer data){
4      while(TRUE){
5          k = k + 1;
6      }
7  }
8  int main(){
9      g_thread_new("thread1", writeToMem, NULL);
10     while (TRUE){
11         printf("int: %i\n", k);
12     }
13 }
```

In diesem Beispiel werden zwei Threads erstellt der erhöht jeweils den Wert einer Zahl im Speicher um 1 und der andere gibt sie aus. In der Praxis sollten solche Aufrufe natürlich kontrollierter und möglichst nich gleichzeitig durchgeführt werden. Die ersten Zeilen der Ausgabe lauten wie folgt:

```
int: 20
int: 21533
int: 23799
int: 25686
int: 27514
int: 29271
```

```
int: 31025
int: 32756
int: 34493
int: 36211
int: 37931
int: 39645
int: 41374
```

Nicht jede Zahl ab 20 wird ausgegeben, was daran liegt das wenn der Vorgang des Erhöhens wesentlich schneller ist als der des Ausgebens, eine Zahl erhöht sich also mehrmals bevor sie ausgegeben wird.
Man erkennt hier sofort, das die beiden while-Schleifen parallel ausgeführt werden.
Ein weiteres interessantes Detail ist das die GLib zu internen Zwecken automatisch einen Thread erstellt, wenn also keine der Thread Funktion verwendet wird kann trotzdem ein Thread entstehen.

2.7 Weitere Funktionen

2.7.1 GObject

Ein weiterer Teil der GLib, ist GObject. Gobject bietet ein Objektsystem in C. Dabei werden OBjekte einfach als structs mit Feldern und Methoden gespeichert. Somit wird Objektorientierte Programmierung in C möglich. GObject wäre allerdings ein Thema für sich und wird daher hier nicht weiter behandelt.

2.7.2 Utilities

Die sogenannten Utilities fassen einige der nützlichen „Hilfsfunktionen" der GLib zusammen.

- Zeit und Datum
- Parser
- Testing
- Regular Expressions

Auch hier bietet die GLib also eine Vielzahl an Möglichkeiten um dem Programmierer das Leben einfacher zu machen.

3 Vorteile und Nachteile

Wann sollten man nun die GLib verwenden und wann ist sie das falsche Werkzeug? Bei GUI Programmierung für Gnome und mit GTK ist die GLib wohl unbestritten eine gute Wahl. Dabei hat man dann zusätzliche Vorteile wie einfaches Crossplatform Entwicklung.
Ist der einzige Grund für die Verwendung der GLib der, das man nur auf einzelne Funktionen zugreifen will sollte man sich wohl überlegen ob man nicht mit einer kleiner, spezialisierten Library zurechtkommt, auch wenn wohl auf den meisten heutigen Systemen der relativ geringe Overhead nicht stören sollte. Hier bietet sich z.B. die ulib [6] schnelle Implementation für Hashtables und Anderes.

4 Quellen

Literatur

[1] „GLib Reference Manual" https://developer.gnome.org/glib/stable/

[2] „GLib - Wikipedia, the free encyclopedia" https://en.wikipedia.org/wiki/GLib

[3] „GLib Event Loop" http://docs.huihoo.com/symbian/nokia-symbian3-developers-library-v0.8/GUID-7FD05006-09C1-4EF4-A2EB-AD98C2FA8866.html

[4] „Thread (computing) - Wikipedia, the free encyclopedia" https://en.wikipedia.org/wiki/Thread_(computing)

[5] „Multithread programming" http://docencia.etsit.urjc.es/moodle/pluginfile.php/1668/mod_folde /GNOME/threads.pdf

[6] „ulib - An efficient algorithm library for C and C++. - Google Project Hosting" https://code.google.com/p/ulib/