Program a graphical user interface
GTK+/GDK/GLIB

/home/bl/Desktop/länk till drivrutiner/graphic/GTKGLIB.sxw

GTK+ Gimp ToolKit

A library for writing apps with a graphical interface

- Objektoriented, written in C
- Support a number of language:
  Ada95, C++, eiffel, pascal, perl, python m.m.
- Windowmanaging, buttons, menus, lists etc.
- Plattform independent

GDK Gimp Drawing Kit

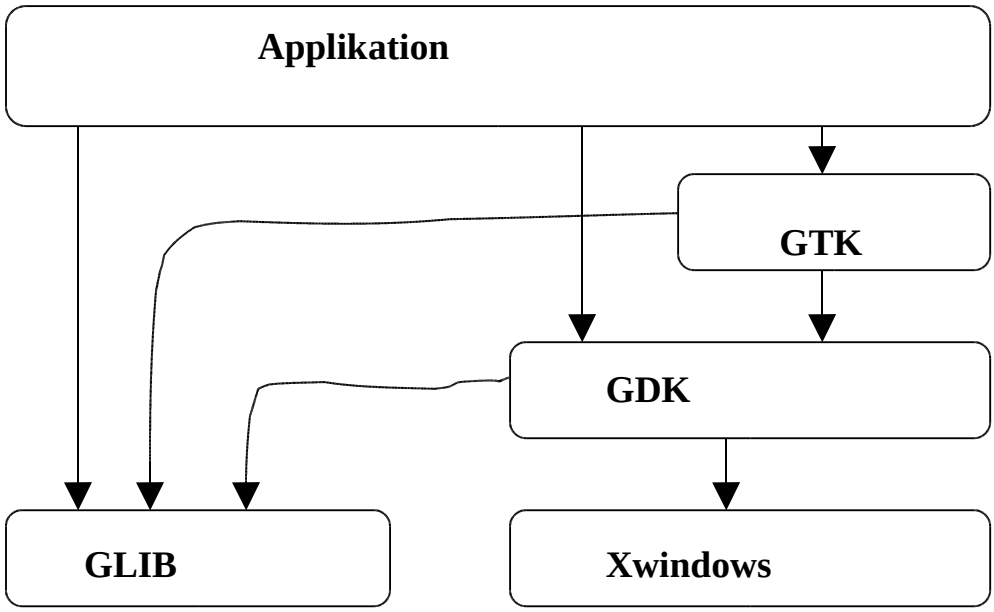
- Builds on Xlib graphic interface.
- Lines, circles, lines, dots, colors etc.
- Plattform dependent

GLib

- Provides datatypes, lists, error management, memory mangement, timers etc.
- Adjusted versions of standard C functions, ie g_string_new, g_print etc

Xlib

Provides libraries for writing to X



And a variety of supporting libraries

pango, gobject, gmodule, math etc

/home/bl/Desktop/länk till drivrutiner/graphic/GTKGLIB.sxw

Development environment

- Make, gcc, gdb
- Kdevelop, Anjuta
- Glade

gcc -Wall -g program.c -o program `pkg-config --cflags --libs gtk+-2.0`

pkg-config returns necessary libpath and includepath etc.

Basic datatypes

- char gchar
- long glong
- int gint
- (void*) gpointer
- osv

- GString
- GList
- etc

Basic structures

Objectoriented model with inheritance -Everything is a GTKObject

- GObject
  - GTKObject
    - GTKWidget
      - GTKContainer
        - GTKBin
          - GTKButton
            - etc

/home/bl/Desktop/länk till drivrutiner/graphic/GTKGLIB.sxw

Creating new objects

1. gtk_*_new() - one of various functions to create a new widget.

2. Connect all signals and events we wish to use to the appropriate handlers.

3. Set the attributes of the widget.

4. Pack the widget into a container using the appropriate call such as gtk_container_add() or gtk_box_pack_start().

5. gtk_widget_show() the widget.

Ex:

```
GtkWidget *my_window, *my_button;
my_window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
my_button = gtk_button_new_with_label(“knapp”);
```

In order to use object-specific operations, the objects must be casted (MACRO´S):

Ex:

```
gtk_container_add(GTK_CONTAINER(my_window),
                                          my_button);
```

All objects must be shown in order to be visible!!!

Ex:

```
gtk_widget_show( my_button);
gtk_widget_show( my_window);
```

Containers

Containers can contain other objects
All objects normally has a parent/child-relation

Ex:

```
GtkWidget *my_window, *my_button;
.
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
my_button = gtk_button_new_with_label ("Hello
World");
gtk_container_add (GTK_CONTAINER (window), button);
```

A minimal program:

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

int main(int argc, char *argv[])
{
        GtkWidget *window;
        gtk_init(&argc, &argv);
        window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
        gtk_widget_show(window);
        gtk_main();
        return 0;
}
```

gtktutorial_1

Signals

- Signal OBS Not the same as lowlevel-signals in Unix/Linux
- Connect every “signal” you want to use to a ”callback-function” and a pointer to the data you want to sent.

```
gulong g_signal_connect( gpointer      *object,
                         const gchar   *name,
                         GCallback     func,
                         gpointer      func_data );
```

```
void callback_func( GtkWidget *widget,
                    ... /* other signal arguments */
                    gpointer   callback_data );
```

Ex:

```
g_signal_connect(GTK_OBJECT(my_button), ”clicked”,
                 GTK_SIGNAL_FUNC(my_function), NULL);
```

```
my_function(GtkWidget *WhereICameFrom, gpointer data)
{
        Do something
        return;
}
```

/home/bl/Desktop/länk till drivrutiner/graphic/GTKGLIB.sxw

Signals for buttons

- pressed
- released
- clicked
- enter
- leave
- etc

Events

Sent by the system

- delete_event ; Signalled by the window manager when a window is going to be destroyed.
- destroy; Signalled when a window is destroyed (if we have connected the window to the signal) or by delete_event-function when returning FALSE.
- etc

Ex:

```
g_signal_connect (GTK_OBJECT (window),"delete_event",
           GTK_SIGNAL_FUNC (delete_event), NULL);

g_signal_connect (GTK_OBJECT (window), "destroy",
                GTK_SIGNAL_FUNC (destroy), NULL);


gint delete_event( GtkWidget *widget,GdkEvent *event,
                   gpointer   data )
{
    g_print ("delete event occurred\n");
      /* Change TRUE to FALSE and the window manager
will emit a "destroy" signal. */
    return(TRUE);
}

void destroy( GtkWidget *widget, gpointer data )
{
    gtk_main_quit();
}
```

Send data to callback-functions

```
static void callback( GtkWidget *widget,
                      gpointer   data )
{
    g_print ("Button %s was pressed\n", (gchar *) data);
}

g_signal_connect (G_OBJECT (button), "clicked",
                  G_CALLBACK (callback), (gpointer) "button 1");
```

gtktutorial_3

A more useful and well-behaved program:

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

void hello( GtkWidget *widget,
            gpointer   data )
{
    g_print ("Hello World\n");
}

gint delete_event( GtkWidget *widget, GdkEvent *event,
gpointer   data )
{
    g_print ("delete event occurred\n");
    return(FALSE);
}

void destroy( GtkWidget *widget, gpointer   data )
{
    gtk_main_quit();
}

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

    gtk_init(&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_signal_connect (GTK_OBJECT (window),
"delete_event",
                        GTK_SIGNAL_FUNC (delete_event),
NULL);

    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC (destroy), NULL);

    button = gtk_button_new_with_label ("Hello World");

    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (hello), NULL);

    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show (button);
    gtk_widget_show (window);
    gtk_main ();
    return(0);
}
```

Boxes

```
GtkWidget *gtk_hbox_new ( gboolean homogeneous,
                          gint     spacing );

GtkWidget *gtk_vbox_new ( gboolean homogeneous,
                          gint     spacing );
```

homogeneous controls whether each object in the box has the same size

A box can contain rows or columns of other objects

```
void gtk_box_pack_start( GtkBox    *box,
                         GtkWidget *child,
                         gboolean   expand,
                         gboolean   fill,
                         guint      padding );

void gtk_box_pack_end( GtkBox    *box,
                         GtkWidget *child,
                         gboolean   expand,
                         gboolean   fill,
                         guint      padding );
```

The **first** argument is the box you are packing the object into
The **second** is the object.
The **expand** argument controls whether the widgets are laid out in the box to fill in all the extra space in the box so the box is expanded to fill the area allotted to it (TRUE); or the box is shrunk to just fit the widgets (FALSE).
The **fill** argument control whether the extra space is allocated to the objects themselves (TRUE), or as extra padding in the box around these objects (FALSE).
Padding is added on either side of an object.

Labels

```
GtkWidget *gtk_label_new( const char *str );
```

gtktutorial_6

Interaction

- Signals/events (buttons)
- Text entrys
- Text widgets

Text Entrys

```
GtkWidget *gtk_entry_new( void );

void gtk_entry_set_text( GtkEntry    *entry,
                         const gchar *text );

const gchar *gtk_entry_get_text( GtkEntry *entry );

void gtk_editable_set_editable( GtkEditable *entry,
                                gboolean     editable
);
```

The function above allows us to toggle the editable state of the Entry widget by passing in a TRUE or FALSE value for the `editable` argument.

```
void gtk_entry_set_visibility( GtkEntry *entry,
                               gboolean  visible );

void gtk_editable_select_region( GtkEditable *entry,
 gint         start,
 gint         end );
```

Ex:

```
gtk_entry_set_text (GTK_ENTRY (entry), "hello");
tmp_pos = GTK_ENTRY (entry)->text_length;
gtk_editable_insert_text (GTK_EDITABLE (entry),
                          " world", -1, &tmp_pos);
gtk_editable_select_region (GTK_EDITABLE (entry),
               0, GTK_ENTRY (entry)->text_length);
```

gtktutorial_7

/home/bl/Desktop/länk till drivrutiner/graphic/GTKGLIB.sxw

More on signals and timers

configure_event
Emitted when drawing area created or resized

expose_event
Emitted when hidden drawing area exposed or when
gtk_widget_queue_draw_area(….);

Timers

```
timerid = gtk_timeout_add(guint time_in_msec,
GtkFunction function, gpointer data )
```

ex:

```
gtk_timeout_add(1000, my_func, drawing_area);

gint my_func(gpointer window)
{
.
}
```

Drawing

- To a drawing area GtkWidget->window / *GdkDrawable
- to a pixmap
- to bitmaps (2 colors)

Drawing area

```
GtkWidget *drawarea;
drawarea = gtk_drawing_area_new();
gtk_drawing_area_size(drawarea, 200, 200);

gdk_draw_rectangle(drawarea->window, …..);
gdk_draw_arc(drawarea->window, …..);

GdkDrawable *drawable;
drawable = drawarea->window;
gdk_draw_rectangle(drawable, …..);
```

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

gint my_draw(gpointer data)
{
      g_print ("In my_draw\n");
      GtkWidget *drawing_area = (GtkWidget *) data;
      gdk_draw_rectangle (drawing_area->window,
                  drawing_area->style->white_gc,TRUE ,
                  0,0, 200,200);
      gdk_draw_line (drawing_area->window,
                  drawing_area->style->black_gc, 0, 0,
                  200, 200);
      return TRUE;
}

int main(int argc, char *argv[])
{
      GtkWidget *window, *drawing_area;
      gtk_init(&argc, &argv);
      window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
      drawing_area = gtk_drawing_area_new();
      gtk_drawing_area_size (GTK_DRAWING_AREA
                  (drawing_area), 200, 200);
      gtk_container_add (GTK_CONTAINER (window),
                  drawing_area);

      gtk_widget_show(drawing_area);
      gtk_widget_show(window);
      gtk_timeout_add ( 1000, my_draw, (gpointer)
                  drawing_area);

      gtk_main();
      return 0;
}
```

drawtutorial_1

/home/bl/Desktop/länk till drivrutiner/graphic/GTKGLIB.sxw

Pixmap

```
GdkPixmap *pixmap;
pixmap = gdk_pixmap_new(widget->window, widget-
>allocation.width, widget->allocation.height, -1);
```

The pixmap inherit some properties from widget->window

```
gdk_draw_rectangle(pixmap, …..);
gdk_draw_line(pixmap, …..);
```

Copy pixmap to a drawing area:

```
gdk_draw_pixmap(dest, graphical contect, src, xsrc,
ysrc, xdest, ydest, width, height);
```

```
gdk_draw_pixmap(widget->window,
     widget->style->fg_gc[GTK_WIDGET_STATE (widget)],
     pixmap,
     event->area.x, event->area.y,
     event->area.x, event->area.y,
     event->area.width,
     event->area.height);
```

drawtutorial_2

An example of how to use the signals/timer above to create a “dubble buffer” system:

configure_event
Free pixmap
Create a new pixmap with new size

expose_event
Copy the newly exposed or updated part of the pixmap to the drawing area

Timer function
Take care of updating the pixmap and eventually call the expose_event function

drawtutorial_3
drawtutorial_4