GTK+ for the Linux Framebuffer

Alexander Larsson
Red Hat, Inc.

alexl@redhat.com

In the upcoming GTK+2.0 release GTK+ will support rendering directly to the framebuffer instead of using the X Window System. This is good for embedded systems such as PDAs and other systems with very limited resources; they are able to run without the overhead of an X server while still taking advantage of the power of GTK+ and the large base of existing programs. This white paper describes how GTK+ for the Linux framebuffer works, its advantages and limitations, how it fits into the environment, and how to obtain the software.

Table of Contents

Introduction

GTK+ is a graphical user interface toolkit that lets you write modern, easy to use applications. It was originally written for use with the X Window System, but the 2.0 release supports several other windowing systems. The port described in this white paper, the framebuffer port (or sometimes just GtkFB), runs without any external windowing system. Instead, the framebuffer port runs straight on the Linux framebuffer device.

A framebuffer is a Linux hardware device that the kernel exports to programs that run in user space, giving those programs access to the graphics card. Having access to the graphics card allows framebuffer device programs to change resolution/bitdepth and read/modify the display memory of the graphics card. Most embedded Linux ports have framebuffer device drivers for their display hardware, and there are drivers for many standard desktop graphics cards. Both of these drivers allow this port of GTK+ to be used on such devices.

GTK+ has traditionally used xlib to make calls over the network to an X server that handles drawing, user input, managing overlapping windows, etc. This client-server architecture is very nice, because it allows several programs to display on the screen at the same time, and because the communication is over the network, they do not even need to be run on the same computer. It is also a good way to structure a large system, because the trusted X server contains a lot of system dependent low-level device handling that other programs should not need to do, and shouldn't even be allowed to do.

The client-server architecture has some overhead though, as all communications must be over the network, and some data is duplicated in the server and the client for performance reasons. Because GtkFB does not use X, it is well suited for environments where the available memory and storage is very low and network transparency is not needed, such as embedded devices.

Overview

GtkFB, just like GTK+, depends on a few basic libraries:

Table 1. Library Dependencies

Library	Description
Glib	**Contains some of the basic data types and functions needed by typical C programs.**
Pango	**Contains support for text and fonts in most languages and scripts.**
libpng, libjpeg, libtiff	**Used by the image loaders. These are optional, and needed only if you wish to load that particular type of image.**
FreeType	**Additional library used by the framebuffer port. Used to render TrueType and Type1 fonts**

GTK+ compiles to three libraries: gdk-pixbuf, gdk, and gtk. Gdk-pixbuf is a library for loading, saving, and manipulating images; gdk is an abstraction layer for the windowing and graphics system; and gtk is the high-level user interface library. Almost all framebuffer specific code is in the gdk library, which is called by the generic gtk code, but there is some framebuffer specific code in gtk to handle managed windows.

The gdk library has three basic tasks: drawing graphics, handling overlapping windows, and generating events. At the core of gdk are the drawing primitives that let you draw lines, polygons, text, and so forth. This is very similar to the X Window system; in fact, a modified version of the X software rasterization library (libmi) is used. Handling overlapping windows means keeping a tree of windows up to date on the screen and making sure that when you draw in one window, you don't draw on overlapping windows. Events are generated when there is user input or when the window system needs to tell the application something, like a window needs repainting, or the mouse moved from one window to another. The events generated are a subset of the ones that X generates, because gtk does not need all the events and event-information that X gives.

When a program linked to the GtkFB libraries starts, it typically calls gtk_init() early in the startup. At this point, GtkFB opens the Linux framebuffer device (and optionally changes to the desired resolution and bit-depth), the keyboard, and the mouse device. It scans a specified set of directories for fonts, and then proceeds to initialize the window and event systems.

GtkFB has built-in drivers for the normal Linux keyboard and several normal mouse types, including touch-screens. If other drivers are needed, it is very easy to add a new driver. GtkFB currently doesn't use any hardware acceleration, but support for hardware acceleration for particular video cards could be added. Currently 8, 16, 24, and 32 bit per pixels framebuffers are supported.

Screenshots[1] of various programs running on a 800x600 16bit framebuffer are available.

History

GTK+ originally comes from the `GIMP` project, where it was developed to write the user interface. It was later split off from the project, and has since been used by many other other programs, including the GNOME desktop project.

The much used GTK+1.2 stable release is based on X, but was ported to Win32 by replacing the gdk library and adding some Win32 support to glib. This support was never merged into the 1.2 release, but because the build system was changed in 2.0, the Win32 port was included.

During the summer of 2000, Elliot Lee of Red Hat started writing the framebuffer port of gdk. I took over the GtkFB work in October 2000 and have been working on it since.

At the release of GTK+2.0, the framebuffer port will be a fully working port of GTK+, with all the features of the X port.

Benefits

The main benefit with GtkFB is that you can use the powerful GTK+ library, with its large base of software, in places where the extra size of X is not wanted, such as PDAs and other embedded devices. The API is also exactly the same as the desktop version, so code can easily be ported and shared between desktop and embedded devices.

Another important benefit of GtkFB is that you are free, and even encouraged, to modify the sourcecode of the library to better fit your needs (in terms of size and features). It is easy to trim GTK+ to fit your particular needs. Due to the LGPL License, you need to release your changes to the library itself, but you don't need to release the source to your own programs.

GtkFB is also free of costs, licensing or otherwise, your own development costs notwithstanding.

Additionally, due to the fact that GtkFB does not use the X protocol, some old limitations of X have been lifted. All text is rendered anti-aliased in 256 *grayscales.* Custom cursors can be any color-depth, and it supports runtime screen rotation.

Limitations

Of course GtkFB has limitations too. The main limitation is the single-process model. All code in the system must be in the same binary and run in the same process. This means you can't use processes to separate and protect different parts of the system from each other. It also makes it harder to design larger systems.

Another problem is that some GTK+ programs make direct X calls when using X features that are not supported in Gdk. These programs cannot be used with GtkFB without change. The GNOME libraries make some direct X calls, so running GNOME programs on the framebuffer may need some work.

X has mature and broad driver support with very good hardware acceleration. GtkFB can support acceleration, but none is currently written, and writing acceleration code can be difficult. This means that GtkFB can be a lot slower, especially on large screens.

Some other interesting X features are not supported by framebuffer, such as network transparency, DGA, multiple screen and visual support, Xv extension, and Xrender extension.

Memory and Storage Requirements

In order to evaluate the memory and storage requirement, I've compiled a small version of GtkFB on an x86 machine. By doing further work on compiler flags and removing features you are not using, you can make the binaries even smaller.

Here are the flags used to build the libraries:

```
glib: ./configure -enable-debug=no -disable-mem-pools
pango: ./configure -enable-debug=no -with-included-modules=yes
gtk+: ./configure -enable-debug=no -with-gdktarget=framebuffer -disable-
shadowfb -disable-modules -with-included-loaders=xpm,png,jpeg
```

This builds both shared and static libraries. For simplicity, I've built xpm, png, and jpeg image loaders into gdk-pixbuf. In a real-world case, you would probably use dynamically loaded image loaders if you chose to use shared libraries.

The stripped, shared GtkFB libraries occupy about 2 MB of disk space. Additionally FreeType is 202KB, libjpeg is 138KB, libpng 126KB, libz (needed by libpng) 58KB.

To give a feel for typical memory usage, I ran the `testgtk` program (included in the GTK+ sources) which shows various widgets. I opened three windows named *button box, buttons,* and *clist* and then analyzed the memory requirements. Estimating memory requirements in a virtual-memory system is a bit hard, because memory can be dynamically paged in and out.

The RSS size (the total amount of physical memory used, not counting pages swapped out) was 3.4MB. In Linux a page from a shared library is not written to swap but just discarded, as it can just be read from the file when needed again. The total virtual memory size was 6.6MB, out of which 2.3MB were shared with other processes. Of

the total virtual size 940KB is the mapped framebuffer, which is not actual RAM memory.

Some further memory statistics: 72KB is the mapped Arial font, 112KB is mapped locale info, 1444KB is mapped libc code, and 120KB is the mapped program binary. The heap is 836KB, and the stack is 24KB.

It is interesting to compare this to a statically linked binary, because when linking statically only the object files actually used by the program are linked into the binary. For comparison, I compiled a version of `testgtk` that has everything linked statically except libc (including libm).

The statically linked binary is 1.8MB, and depends on libc, libm, libdl and ld-linux only. The RSS was slightly lower, at 2.5MB, the total virtual memory size was 5.2MB, and 1.7MB memory was shared. The other statistics from above are the same in the statically linked case.

The `testgtk` program uses a lot of widgets and GTK+ features, which means linking it statically brings in a lot of code. To see what the size of a minimal program can be I compiled the `buttons.c` example distributed with GTK+. It just opens a window with a button containing an image and a label. The binary is 1.3MB, RSS was at 1.9MB and total virtual memory size 4.4MB. The heap is 376KB and the stack 20KB.

If few other programs using libc are used on the system you might want to link it statically too. Fully statically linked binaries (including libc and libm) are 2.1MB for `testgtk` (RSS 2.5MB, VM size 5.4MB) and 1.7MB for buttons.

X Window System Size Comparison

A quick comparison with the X based GTK+ is in order. I compiled an X version using the same flags as the framebuffer version. I ran `testgtk` (using shared libraries) on an Xserver that had only `twm` running.

The `testgtk` total virtual memory size was 6.6MB, and the RSS was 4MB. 980KB of mapped libraries are present that was not in the GtkFB version (X libs and thread libs). The X version of pango is slightly larger, because it has some extra foreign language shapers. The Xserver (XFree86 4.0.1, Matrox server) total virtual memory size is 9.2MB, not counting the mapped video ram. RSS is 7.9MB. Some of the memory is shared with `testgtk` (somewhere around 2MB). Additionally twm has a VM size of 3.2MB and a RSS of 1.6MB, but there are smaller window managers, and you might not even need one.

The gtk and gdk-pixbuf libraries are the same size on disk, but the X gdk library is 48KB smaller. Extra dependencies for the X version are libX11 (824KB), libXext (51KB), and libpthread (451KB, only needed if you use threadsafe X libs). Furthermore, you need an X server. The XFree86 4.0.1 one I used is 1.5 MB, not counting various driver modules, but there are smaller Xservers available (tinyX is about 700KB).

The result of this simple experiment shows that framebuffer GTK+ programs are somewhat smaller than their corresponding X versions, ignoring the size of the Xserver. If you add an X server and a window manager the X version is a lot larger. However, the GTK+ framebuffer port does have small window managing support which makes it ideal for smaller devices where the overhead of X can be too much for available memory.

Does GtkFB Replace X on the Desktop?

Many people think GtkFB will or should replace X on the desktop. This is a misunderstanding; GtkFB is not really meant to run on the desktop (except for specific

cases like distribution installers). For quite a few reasons outlined in the Limitations section above, GtkFB is inferior to X on a typical desktop.

A possible cause of this thinking is the common misunderstanding that the X Window System is a big, bloated, slow program that doesn't do much good. The truth of the matter is that X servers often look huge when you see them in `top`, but this is due to the fact that they have mmap'ed the whole graphics card memory, and they keep a lot of pixmap data that really belongs to other processes. Because X is network independent, it can never be as efficient as an optimal non-network independent, windowing system, but the performance loss is actually very small, and the gains are tremendous. The current problems with X (font handling, no Anti Aliasing, bad 3D hardware support, no video support), are currently being fixed by X extensions.

Source

Source releases of GTK+ 2.0 betas (named 1.3.x) can be found on the GTK+ FTP site[2]. Version 1.3.2 (not released at the time of writing) will be a good release to use if you want to try GtkFB.

In addition to glib, pango, and GTK+ you need FreeType (2.0.1 or later), which can be found on the FreeType FTP[3] site. Optionally, to build the corresponding gdk-pixbuf loader, you can use libjpeg, libpng, and libtiff.

GtkFB specific build documentation is in `gtk+/docs/README.linux-fb`.

The latest version can always be found in the GNOME `CVS` tree. You will need to check out and build these modules (in this order): glib, pango, and gtk+. The GNOME Developer's Website[4] contains a description[5] of how to check out modules from the GNOME `CVS` tree.

Notes

1. http://people.redhat.com/~alexl/gtkfbscreenshots.html
2. ftp://ftp.gtk.org/pub/gtk/v1.3/
3. ftp://ftp.freetype.org/pub/freetype2/
4. http://developer.gnome.org
5. http://developer.gnome.org/tools/cvs.html