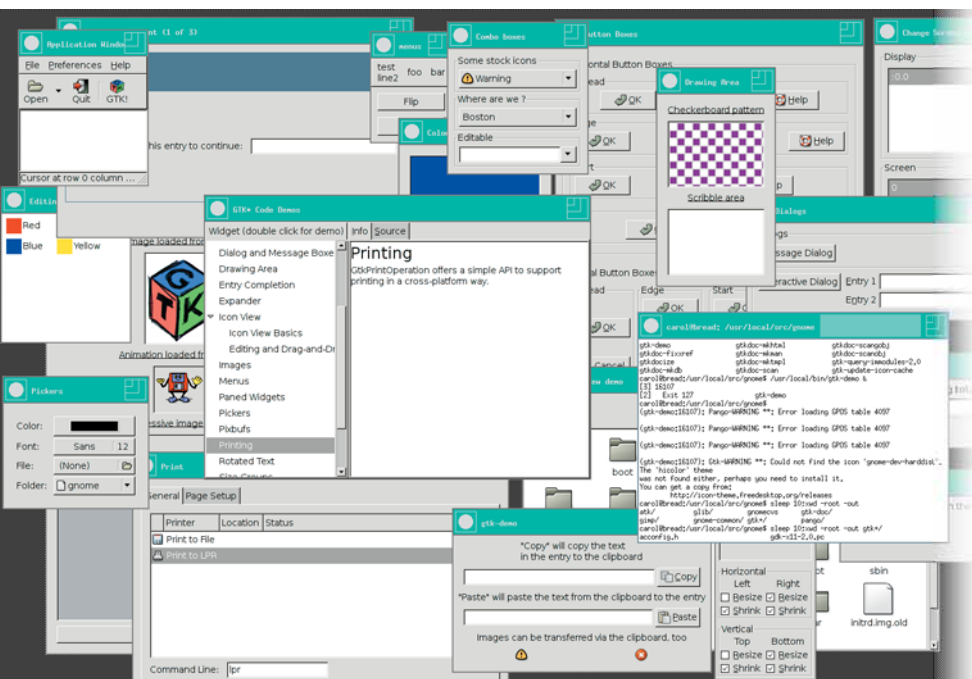




Новая серия! Разработка переносимых приложений с графическим интерфейсом пользователя

GTK+ первое

ЧАСТЬ 1: Прочитав нашу серию статей о Qt/KDE, вы укрепились во мнении, что о лучшем и мечтать нельзя, и уже готовы удивить мир своей разработкой? Выслушайте сначала контр-аргументы – **Андрей Боровский** предлагает вашему вниманию новую серию учебников о GTK+!



Вопреки распространенному убеждению, эти библиотеки можно использовать в коммерческих проектах. Даже Билл Гейтс смог бы воспользоваться ими!

Руководство программиста FLTK 1.1.7

то в вашей системе наверняка также установлена программа *Devhelp*, которая представляет собой браузер документации программиста по GTK+, GNOME, Gimp и Evolution.

Можно ли назвать GTK+ предпочтительным инструментарием разработчика графических приложений? Вообще говоря, если вы выбираете набор инструментов для создания графического интерфейса нового Linux-приложения, вы не должны особенно беспокоиться о том, какую оболочку предпочитают ваши пользователи. В соответствии с идеологией свободного выбора, которая пронизывает Linux, различные платформы и компоненты системы очень хорошо уживаются между собой. Средства взаимодействия между приложениями, использующими разные графические компоненты, тоже быстро совершенствуются. Для того чтобы сделать выбор между GTK+ и Qt/KDE, следует хотя бы бегло ознакомиться с возможностями и особенностями каждого инструментария.

Как это нередко бывает в мире открытого ПО, в настоящее время активно используются сразу две ветки GTK+. Многие разработчики приложений, воспользовавшиеся в свое время GTK+ 1.2, не видят необходимости переходить на новые версии пакета, поэтому GTK+ 1.2 все еще можно встретить во многих дистрибутивах Linux. Новые (на момент написания этой статьи) приложения используют GTK+ версий 2.x. Примеры из нашей серии статей должны работать со всеми версиями GTK+, начиная с 2.0, если иное не указано явно.

Наши инструменты

Приложения GTK+ под Linux – это, прежде всего, приложения Linux. Для создания приложений GTK+ вам понадобятся стандартные инструменты разработчика – GCC и automake со товарищи. Помимо этого, в вашей системе должны быть установлены пакеты GTK+-devel*, atk-devel*, pango-devel* со всеми зависимостями. Кроме того, рекомендуем установить пакеты libgnome-devel и glade. После установки пакетов разработчика в нашем распоряжении окажутся утилиты командной строки glib-config и pkg-config (в некоторых системах также может быть установлена утилита GTK-config). Эти утилиты выводят информацию о расположении базовых библиотек GTK+ в вашей системе. При этом утилита glib-config предназначена исключительно для вывода информации о библиотеке glib, а pkg-config выводит информацию о самых разных библиотеках. Если, например, скопировать в окне консоли

```
pkg-config --libs GTK+-2.0
```

вы увидите нечто вроде

```
-L/usr/X11R6/lib -L/opt/gnome/lib -lGTK-x11-2.0 -lgdk-x11-2.0 -latk-1.0 -lgdk_pixbuf-2.0
```

Линус любит KDE. Он сам неоднократно говорил об этом, не стесняясь в выражениях по адресу GNOME. Я тоже не большой поклонник GNOME, а вот инструментарий GTK+ мне очень нравится. Думаю, в таком сочетании предпочтений нет ничего особенного, ведь GTK+ – это далеко не только GNOME. Даже если вы – фанатичный пользователь KDE, вы наверняка время от времени работаете в редакторе Gimp, основанном на библиотеках GTK+. Собственно, аббревиатура GTK+ и расшифровывается как Gimp ToolKit – «набор инструментов для Gimp».

Сегодня мы начинаем серию статей, посвященных GTK+. Большая часть приложений, которые мы напишем, будет работать независимо от среды GNOME, но мы не оставим без внимания и взаимодействие с этой популярной оболочкой. Скорее всего, вы, уважаемый читатель Linux Format, уже представляете себе, что такое GTK+. Тем не менее, следует все-таки напомнить, что GTK+ представляет собой открытый набор графических компонентов, предназначенных для создания приложений на платформах Linux/Unix, Win32 и MacOS. GTK+ включает в себя большое количество визуальных элементов, используемых при создании графического интерфейса приложений, а также некоторые вспомогательные невидимые элементы. На GTK+ основаны такие приложения, как графическая среда GNOME, редактор растровой графики Gimp, текстовый редактор AbiWord, табличный процессор Gnumeric и многие-многие другие. Помимо библиотек, реализующих различные элементы графического интерфейса приложений, GTK+ снабжен вспомогательными утилитами. Glade, например, позволяет проектировать интерфейсы GTK-приложений в режиме визуального редактирования. Если вы установили пакеты разработки GTK+/GNOME,



ЗНАКОМСТВО

```
-lpangocairo-1.0 -lpango-1.0 -lcairo -lobject-2.0 -lgmodule-2.0 -ldl -lglib-2.0
```

```
-lfreetype -lfontconfig -lXrender -lX11 -lXext -lpng12 -lz -lglib-2.0 -lm
```

Вывод команды представляет собой список ключей компоновщика GCC, которые необходимо указать для подключения к основанному на GTK+ 2.x приложению всех необходимых ему библиотек. Как вы, конечно, догадались, этот список сгенерирован не столько для того, чтобы удовлетворить наше любопытство, сколько для автоматизации работы компоновщика в процессе сборки приложений GTK+, что мы увидим ниже. Команда

```
pkg-config --cflags GTK+-2.0
```

выдаст все ключи компилятора, необходимые для компиляции приложения GTK+ 2.x.

Hello GTK+ World!

Теперь, когда мы знаем, что нам нужно для того, чтобы скомпилировать приложение GTK+, мы готовы написать простейшую программу. Наша первая программа (назовем ее *helloworld*) создает простое окно с кнопкой. Щелчок по кнопке приводит к закрытию окна и завершению работы программы. Рассмотрим исходный текст программы *helloworld* (этот исходный текст вы найдете на диске в файле *helloworld.c*).

```
#include <GTK/GTK.h>
static void button_clicked(GtkWidget * widget, gpointer data)
{
    g_print("Button was clicked!\n");
}
static gboolean delete_event(GtkWidget * widget, GdkEvent * event,
```

»

GTK+ или Qt/KDE?

Сделать выбор между двумя популярными графическими инструментами, которые долгое время конкурировали друг с другом и потому многое друг у друга переняли, непросто. Критерием истины здесь может быть только практика разработки вашего проекта, однако, мы попробуем провести некоторый формальный анализ. Сводка важнейших параметров GTK+ и Qt приводится в следующей таблице.

Функция	GTK+	Qt
Базовый интерфейс	C	C++
Лицензия	LGPL	Двойная
Порт для Win32 и MacOS	+	+
Возможность прямых вызовов из C	+	-
Интерфейсы Java, Perl, Python	+	+
Порт для .NET	+(GTK#)	+(Qt#)
Бесплатно для коммерческого использования	+	-

То, что по многим параметрам и GTK+, и Qt выставлены плюсы, не означает, что соответствующие возможности этих платформ равноценны. Например, порты GTK# и Qt# на данный момент отличаются довольно сильно: первый проект поддерживается командой разработчиков Mono и, ввиду незавершенности Mono-реализации Windows.Forms, является в нем де-факто стандартом для создания GUI; второй является любительским и не видел обновлений аж с 2002 года! С Java ситуация в какой-то мере обратная: привязки Qt Jambi разрабатываются непосредственно Trolltech, тогда как интеграция GTK+ и Java выполняется в рамках стороннего проекта Java-GNOME, который хоть и включает в себя интерфейсы Java для всех базовых компонентов GTK+, но ориентирован прежде всего на GNOME. Python одинаково хорошо работает и с Qt, и GTK+, а PerlQt не обновлялся с 2003 года и не поддерживает Qt4. В целом, можно сказать, что GTK+ выигрывает в поддержке «неродных» и скриптовых языков — по крайней мере, «по очкам».

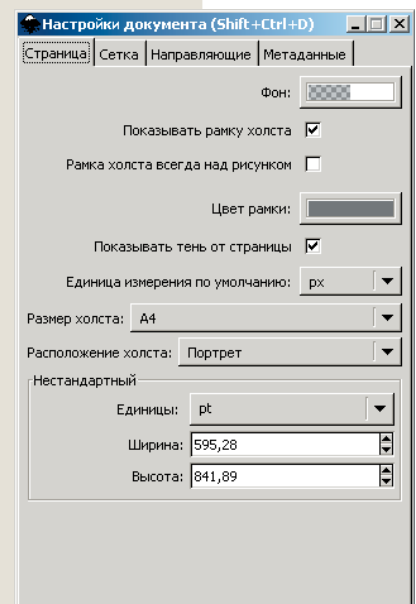
Остановимся на том, что важно для любого программиста — на лицензировании. GTK+ распространяется на условиях LGPL. Иначе говоря, его можно использовать совершенно бесплатно и для создания открытых приложений, и для коммерческих разработок (код приложения может быть закрытым, если он не является расширением самого набора GTK+). С Qt ситуация иная. Вы можете пользоваться Qt бесплатно для создания открытых программ, но за лицензию, позволяющую

разрабатывать коммерческие продукты, придется платить, причем немало.

С кросс-платформенностью тоже не все так просто, как может показаться на первый взгляд. Большинство Qt-ориентированных Linux-проектов использует не только Qt, но и дополнительную функциональность, которую предоставляют библиотеки KDE. В то же время в тандеме Qt/KDE по-настоящему кросс-платформенной является только Qt (ситуация изменится после выхода KDE4, но последняя не доросла еще даже до бета-версии). Из этого следует, что если вы намереваетесь создавать кросс-платформенный продукт и делаете свой выбор в пользу Qt/KDE, вам придется ограничиться возможностями Qt. В пользу «сладкой парочки» Qt/KDE следует сказать, однако, что, по мнению многих разработчиков, основанный на C++ интерфейс Qt/KDE проще и компактнее, чем ориентированный на C интерфейс GTK+. Кроме того, Qt гораздо аккуратнее интегрирует ваше приложение с окружающей средой: Skype в Windows выглядит и ведет себя в точности как приложение Windows, и немногие догадывались, что Google Earth использует Qt (а не, скажем, MFC) до выхода Linux-версии. Приложения GTK+ тоже можно собрать и запустить в Windows и Mac OS — но не ждите особой эстетики и безупречного поведения.

Важно также понимать, что как Qt/KDE, так и GTK+ используют при построении интерфейса принципы объектно-ориентированного программирования. Разница между двумя наборами инструментов заключается в том, что объектно-ориентированная модель GTK+ реализована «добровольно» в рамках интерфейса C, тогда как интерфейс программирования Qt/KDE закован в объектно-ориентированные конструкции C++. Функции интерфейса GTK+ могут быть вызваны из программ, написанных на языках, которые не поддерживают импорт классов C++ в формате GCC. Для того, чтобы не объектно-ориентированные языки могли импортировать интерфейс Qt/KDE, приходится создавать громоздкие комплексы функций-обертки, «переводящих» вызовы методов классов C++ в формат C.

► Угловатые, «неродные» виджеты не дают Inkscape почувствовать себя в Windows, как дома.



»



```

gpointer data)
{
    g_print("Delete event occurred\n");
    return FALSE;
}

static void destroy(GtkWidget * widget, gpointer data)
{
    g_print("Destroy signal was sent\n");
    GTK_main_quit();
}

int main(int argc, char ** argv)
{
    GtkWidget * window;
    GtkWidget * button;
    const gchar * title = "Hello World!";
    GTK_init(&argc, &argv);
    window = GTK_window_new(GTK_WINDOW_TOPLEVEL);
    GTK_window_set_title(GTK_WINDOW(window), title);
    GTK_container_set_border_width(GTK_CONTAINER(window), 10);
    g_signal_connect(G_OBJECT(window), "delete_event",
        G_CALLBACK(delete_event), NULL);
    g_signal_connect(G_OBJECT(window), "destroy",
        G_CALLBACK(destroy), NULL);
    button = GTK_button_new_with_label("Quit");
    g_signal_connect(G_OBJECT(button), "clicked",
        G_CALLBACK(button_clicked), NULL);
    g_signal_connect_swapped(G_OBJECT(button), "clicked",
        G_CALLBACK(GTK_widget_destroy), G_OBJECT(window));
    GTK_container_add(GTK_CONTAINER(window), button);
    GTK_widget_show(button);
    GTK_widget_show(window);
    GTK_main();
    return 0;
}

```

Прежде чем мы пройдем нашу программу шаг за шагом, необходимо дать некоторые пояснения общего характера. В графических интерфейсах, реализованных на объектно-ориентированных языках, все визуальные элементы представляются классами. В интерфейсе GTK+, реализованном на языке C, визуальным элементам соответствуют структуры данных. Эти структуры (мы будем называть их объектами) группируются в иерархии, которые соответствуют отношениям объектов интерфейса. Следует отметить, что понятие «иерархии» здесь достаточно условное, отношения объектов GTK+ не следует путать с иерархическими отношениями классов в объектно-ориентированных языках.

Корнем иерархии объектов GTK+ является абстрактный объект **GObject**. Ниже в иерархической лестнице расположен объект **GTKObject**, потомком которого является объект **GtkWidget**, который, в свою очередь, служит корнем иерархии всех визуальных элементов (виджетов). Здесь уместно сказать несколько слов и о формировании идентификаторов в GTK+. Как и во многих интерфейсах Unix, имена функций, типов данных, констант и макросов в GTK+ начинаются с префикс-

са, указывающего на имя библиотеки, которая экспортирует данный идентификатор. Имена функций, экспортируемых библиотекой **GTK** (**libGTK**), начинаются с префикса **gtk_**, имена типов данных из этой библиотеки предваряются префиксом **GTK_**, а имена констант и макросов имеют префикс **GTK_**. Имена функций, типов и констант с макросами, экспортируемых библиотекой **GLib** (**libglib**), начинаются с префиксов **g_**, **g** и **G_**, соответственно.

Текст нашей программы *helloworld* начинается с определения трех статических функций. Эти функции представляют собой обработчики сигналов GTK+. Как и все современные многооконные графические системы, GTK+ базируется на событийно-управляемой архитектуре. Когда в графической системе происходит нечто, связанное с одним из окон приложения (щелчок мышью, нажатие на клавишу, сокрытие окном другого приложения – то есть возникает необходимость перерисовки), данному окну посылается сообщение. GTK+ преобразует сообщение оконной системы в сигнал и вызывает функцию-обработчик этого сигнала. В качестве аргументов функции-обработчику передаются данные об объекте-источнике и параметрах события. Механизм сигналов абстрагирован от механизма сообщений низкоуровневой графической подсистемы и отражает скорее структуру GTK+, нежели структуру системы низкого уровня. Источником сигналов, связанных с визуальным элементом управления, в GTK+ считается сам визуальный элемент. Для обработки сигналов GTK+ использует функции обратного вызова (callback). Qt, кстати, тоже использует функции обратного вызова, скрытые под надстройкой сигналов и слотов. Похоже, что прогрессивное человечество, по крайней мере, та его часть, которая пишет на C и C++, ничего лучшего пока не придумало. Обработчики сигналов представляют собой обычные функции C. Например, если мы создадим функцию-обработчик **button_clicked** и свяжем ее с сигналом **clicked** визуального элемента-кнопки, обработчик **button_clicked** будет вызываться в ответ на щелчок мышью по кнопке. Мы можем связать один обработчик с несколькими сигналами и назначить одному сигналу несколько обработчиков. Первым параметром функции обработчика должен быть указатель на объект-источник сигнала, вторым параметром – указатель на произвольную структуру данных, которую программист может связать с данным сигналом. Помимо сигналов и GTK+ определены события, которые соответствуют событиям низкоуровневой системы X Window. По сути, события – это те же сигналы, но функции-обработчики событий отличаются от обработчиков обычных сигналов списком параметров. Имена событий имеют окончание **_event**. В нашей программе функция **button_clicked()** является обработчиком сигнала, а функция **delete_event()** – обработчиком события. Вы можете видеть, что списки параметров этих функций различаются.

Теперь мы можем более подробно описать каждую функцию обратного вызова, определенную в нашей программе. Функция **button_clicked** – это обработчик сигнала **clicked** кнопки приложения. Функция **delete_event** обрабатывает событие **delete_event**, а функция **destroy** представляет собой обработчик сигнала **destroy**. Событие **delete_event** генерируется системой X Window в случае, если пользователь пытается закрыть окно приложения. Сигнал **destroy** всегда посылается приложению GTK+ во время завершения его работы. Действия, выполняемые функцией **main()** нашей программы, можно разделить на шесть стадий: создание и настройка главного окна, назначение обработчиков сигналов и событий окна, создание кнопки, назначение обработчиков сигнала **clicked** кнопки, расположение кнопки в главном окне, отображение окна и кнопки.

Работа программы начинается с вызова функции **gtk_init()**. Как следует из названия, **gtk_init()** инициализирует приложение (устанавливает значения параметров GTK+) и обрабатывает специальные аргументы командной строки, которые могут быть переданы приложению GTK+.

Далее мы создаем главное окно приложения с помощью функции **gtk_window_new()**. Единственный параметр функции указывает, что мы создаем обычное главное окно. Другой возможный параметр – **GTK_WINDOW_POPUP** позволяет создать всплывающее окно (popup window). Функция **gtk_window_new()** возвращает указатель на структуру **GtkWidget**, соответствующую созданному окну. Этот указатель

мы сохраняем в переменной `window`. Функция `gtk_window_set_title()` устанавливает надпись в заголовке окна. Первым аргументом функции должен быть идентификатор окна, вторым аргументом – текст заголовка. Функция `gtk_window_set_title()` ожидает, что первым аргументом будет значение типа `GTKWindow *` (указатель на объект-окно). Однако, поскольку переменная `window` имеет тип `указатель на GtkWidget`, мы выполняем приведение типа с помощью макроса `GTK_WINDOW`. Приведение типов в данном случае необязательно, и мы выполняем его только для того, чтобы избавиться от предупреждений, выдаваемых компилятором. Главное окно приложения, как и многие другие объекты `GTK+`, представляет собой контейнер. Объекты-контейнеры могут содержать произвольное количество дочерних визуальных элементов. При этом контейнеры обычно управляют расположением и размером дочерних виджетов, чем очень облегчают жизнь программиста (более подробное знакомство с контейнерами состоится в одной из следующих статей серии). Функция `gtk_container_set_border_width()` устанавливает ширину границы контейнера. Для главного окна «ширина границы» означает расстояние от края дочернего элемента до края окна.

Функция `g_signal_connect()` связывает сигнал объекта с обработчиком. Первый параметр функции – объект-источник сигнала. Мы приводим идентификатор объекта к типу `GObject` с помощью макроса `G_OBJECT`. Второй параметр функции – строка с именем сигнала. Третий параметр – это указатель на функцию-обработчик (мы приводим его значение к типу `GCallback`). Последний параметр функции `g_signal_connect()` представляет собой указатель на произвольную структуру данных, которую мы можем передать обработчику события. Для большинства обработчиков событий мы будем оставлять это значение равным `null`. Мы назначаем обработчики двум другим сигналам окна – `delete_event` и `destroy` (для удобства имена функций-обработчиков выбраны аналогичными именам сигнала).

Теперь нам необходимо создать кнопку, для чего мы пользуемся функцией `gtk_button_new_with_label()`. С помощью этой функции, чье имя говорит само за себя, мы создаем кнопку и указываем надпись на ней. Единственный параметр функции `gtk_button_new_with_label()` – это текст надписи на кнопке. Возвращаемое функцией значение представляет собой указатель на объект `GtkWidget`, соответствующий созданной кнопке. Почему функции, создающие окно и кнопку, возвращают указатели на объекты `GtkWidget`, а не на соответствующие этим визуальным элементам объекты `GTKWindow` и `GTKButton` (оба типа определены в библиотеке `GTK`)? Вероятно, это сделано для того, чтобы избавить нас от лишних операций приведения типов. Ведь большая часть функций, работающих с визуальными элементами, принимает в качестве параметра именно указатель на `GtkWidget`.

Кнопка может быть источником нескольких сигналов, важнейший из которых, сигнал `clicked`, оповещает приложение о том, что по кнопке щелкнули мышью (или выполнили аналогичное действие с помощью клавиатуры). Мы назначаем сигналу `clicked` кнопки `button` два обработчика событий – определенный нами обработчик `button_clicked()` и функцию `gtk_widget_destroy()`, которая завершает работу программы, посылая при этом сигнал `destroy`. Любопытно отметить, что функция `gtk_widget_destroy()`, вообще говоря, не является обработчиком сигнала. Возможность использовать «простые» функции `GTK+` в качестве обработчиков сигналов представляет собой полезную особенность API `GTK+`, с которой мы еще встретимся не один раз. Для назначения обработчика `gtk_widget_destroy()` мы используем функцию `g_signal_connect_swapped()`. Эта функция аналогична функции `g_signal_connect()`, за исключением того, что при вызове обработчика параметры передаются ему в другом порядке (подробнее об использовании `g_signal_connect_swapped()` мы поговорим в следующих статьях).

Функция `gtk_container_add()` добавляет дочерний элемент в контейнер. Первый параметр функции – это указатель на объект-контейнер, вторым параметром должен быть указатель на добавляемый объект.

Мы используем функцию `gtk_container_add()` для того, чтобы разместить кнопку в окне. И кнопка, и окно примут при этом разумные размеры (определяемые длиной надписи на кнопке и шириной границы окна-контейнера). Наконец, мы выполняем два вызова `gtk_widget_show()`, делающих визуальные элементы интерфейса (кнопку и окно) видимыми. Последняя функция, которую мы вызываем явным образом – `gtk_main()`, она запускает цикл обработки сообщений. Теперь наша программа сможет реагировать на сигналы, посылаемые визуальными элементами управления.

Нам осталось рассмотреть обработчики сигналов. Обработчик `button_clicked()` распечатывает в окне терминала сообщение о том, что кнопка была нажата. Для этого используется функция `g_print()`, которая работает аналогично функции `printf()`. Сигнал-событие `delete_event` посылается программе, как уже говорилось, при попытке закрыть окно приложения. Обработчик этого события может отменить завершение работы приложения (для этого, как ни странно, он должен вернуть значение `true`, а не `false`). Сигнал `destroy` сообщает приложению, что его работа будет завершена и не допускает отмены завершения. Если вы закрываете окно приложения, щелкая по кнопке в заголовке окна, приложение сначала получит сигнал `delete_event`, а затем, если обработчик этого сигнала вернет значение `FALSE`, сигнал `destroy`. Функция `gtk_widget_destroy()` (связанная в нашем примере с кнопкой `button`) посылает только сигнал `destroy`, но не `delete_event`. Хотя получение сигнала `destroy` в нашей программе свидетельствует об уничтожении главного визуального элемента (и его дочерних элементов), само по себе это уничтожение не приводит к выходу из цикла обработки сообщений. Выход из цикла нужно выполнить явным образом с помощью функции `gtk_main_quit()`.

Для того, чтобы скомпилировать наш пример, мы воспользуемся уже известной нам утилитой `pkg-config`. Команда компиляции может выглядеть так:

```
gcc -Wall helloworld.c -o helloworld `pkg-config --cflags GTK+-2.0`
`pkg-config --libs GTK+-2.0`
```

Мы вставляем в командную строку ключи, которые выдает нам утилита `pkg-config`. Теперь мы можем, наконец, полюбоваться творением наших рук (рис. 1).

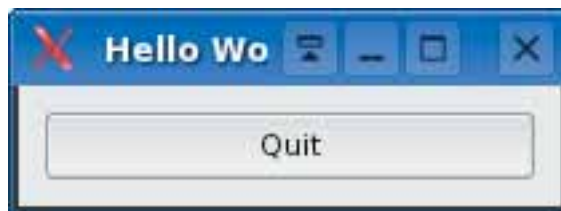


Рисунок 1. Наше первое приложение GTK+.

Приступая к изучению нового интерфейса, очень приятно быстро получить первую работающую программу. Еще приятнее понимать, как это программа работает. Углубленному изучению механизма сигналов и взаимодействия визуальных компонентов `GTK+` будет посвящена следующая статья. [LXF](#)