摘要

我把毕业设计所做的程序命名为 wdl。本文将详细解释 wdl 的工作原理，包括工作流程、多线程控制、任务管理等。

wdl 由我本人单独设计和实现，对 wdl 本身或者本文档有任何疑问或建议，你都可以联系我 Wiky L。

wdl 的源代码托管在 Launchpad，使用 GPLv3 授权。

1. 介绍

wdl 是一个下载管理器，支持 FTP、HTTP(S)和 BitTorrent 协议，多进程多任务下载。完全采用 C 语言实现。wdl 的实现是为了体现 socket 文件传输，wdl 主要依赖以下第三方的库：GTK+、libcurl 和 libtransmission。虽然 wdl 本身没有任何 socket 的调用，但所依赖的库实际上采用的是 socket 通信。在后面本文也将详细解释其实现原理。

GTK+是一个 C 语言实现的用于创建用户界面的图形库。最初是为了开发 GNU 图像处理程序（GIMP）而设计的。现在它已是 linux 系统下最流行的图形库了，GNOME 就是基于 GTK+开发的。

libcurl 可以被解释为 cURL，即 C 语言实现的 URL 传输库。它支持 FTP、FTPS、Gopher、HTTP(S)、SCP、SFTP、TFTP、Telnet 等文件传输协议。在 wdl 中只取其中的 FTP 和 HTTP(S)。libcurl 是线程安全的，兼容 IPv6。

libtransmission 是 linux 系统下著名的 BT 客户端程序 transmission 所采用的 BT 协议库。libtransmission 没有被单独发布，因而也缺少文档支持，我从 transmission 的源代码中将其提取出来。

1.1 面向的读者

本文主要为了毕业设计而写，不过任何对 wdl 工作原理感兴趣的人，或者想借此学习一些相关知识的人也将受益。

1.2 约定

BT：即 BitTorrent 协议。

大小写：为了书写方便，很多专有名词都是全大写或者全小写，比如 GTK+，GLIB 等；但读者应该清楚，除非有特殊说明，Gtk+或者 GLib 具有相同的意思。

源代码：当我说源代码路径时，都是相对于库的源代码根目录。比如，在 GLIB 章节说到源代码 gib/gmain.c:3858 表示 GLIB 源代码目录下，glib 目录中 gmain.c 文件中第 3858 行。注意，可能因为版本不同造成不一致。在本文中，相应的源代码版本分别为 glib2.0-2.38.1、gtk+3.0-3.8.6、curl-7.32.0、transmission-2.82。

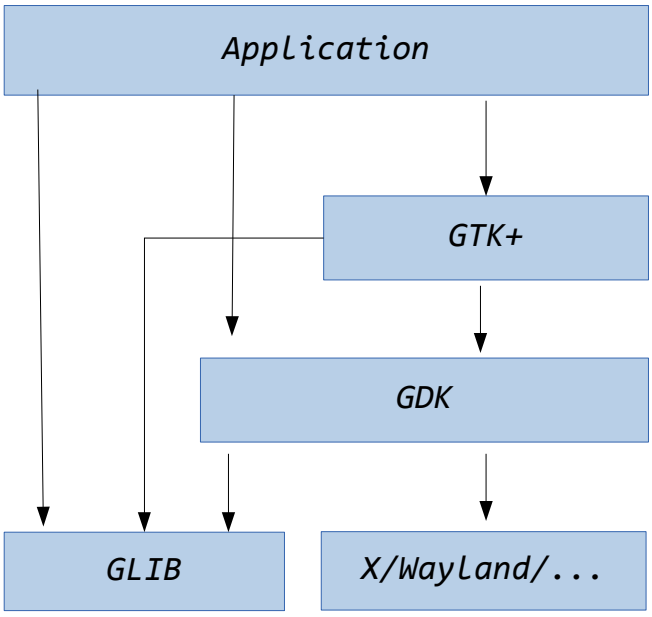
链接：本文中包含很多外部链接，这些链接我无法保证一直有效。

2. 第三方库

wdl 中主要采用了 GTK+、libcurl 和 libtransmission 三个第三方库。本章将详细介绍这三个库在 wdl 中起到的作用。

2.1 GTK+

首先看一下一个典型的 GTK+应用程序的结构。



如图所示，GTK 主要基于 GLIB 和 GDK 两个框架。GDK 是实现图形渲染的引擎，负责具体的界面显示，虽然直接影响了 wdl 的界面样式，但与 wdl 的功能实现关系很少；本文将不对 GDK 做具体讨论，有兴趣的读者可以参考 GDK 的官方文档。

不仅 GTK+本身依赖于 GLIB，wdl 本身也依赖与 GLIB。本节将详细解释 GLIB 如何在 GTK+中起作用，关于 wdl 如何使用 GLIB 的内容在 wdl 的实现中具体讨论。

2.1.1 GTK+命名规范

在使用 GTK+时，了解其命名规范是很有必要的。GTK+的命名规范其实是 GNOME 定义的。详细可以参考官方文档。本节做简单介绍。

普通类型名：全小写，以'g'开头，比如 gint，gchar。

类名：驼峰写法，首字母大写，比如 GtkWindow,GdkEvent。

函数名：小写夹下划线写法，以如 gtk_main,gtk_window_new

常数：大写夹下划线写法，比如 GTK_WINDOW_TOPLEVEL,GTK_ORIENTATION_HORIZONTAL。

2.1.2 GLIB

GLIB 不是 GLIBC，两者有时会被混淆。GLIBC 指的是 GNU C library，有时也被称为 gnulib。

GLIB 最初是 GTK+的一部分，自 GTK+2.0 以后，开发者认为可以将 GTK+中与 GUI 无关的部分独立出来，于是便有了 GLIB。GLIB 主要有 5 部分组成，分别是 GObject、Glib、GModule、GThread 和 GIO。

GObject 是指 GLib Object System，提供了 C 语言以及跨语言的面向对象框架。GTK+采用 GObject 面向对象的方式组织，一个典型的 GTK+类继承树如下。

```
GObject
    GInitiallyUnowned
        GtkWidget
            GtkContainer
                GtkBin
                    GtkWindow
                        GtkDialog
                            GtkAboutDialog
                            GtkAppChooserDialog
                            GtkColorChooserDialog
                            GtkColorSelectionDialog
                            GtkFileChooserDialog
                            GtkFontChooserDialog
                            GtkFontSelectionDialog
                            GtkMessageDialog
                            GtkPageSetupUnixDialog
                            GtkPrintUnixDialog
                            GtkRecentChooserDialog
                        GtkApplicationWindow
                        GtkAssistant
                        GtkOffscreenWindow
                        GtkPlug
                    GtkAlignment
                    GtkComboBox
                        GtkAppChooserButton
                        GtkComboBoxText
                    GtkFrame
```

对于不了解 GObject 的读者，可以简单地将其理解为 C++中类提供的特性，虽然两者还是有很大差别。毕竟面向对象仅仅是一种软件组织方式。wdl 本身也采用 GObject 的面向对象框架。

Glib（这里需要注意大小写）包含了一些通用的处理函数，字符串处理，内存管理等。其中比较重要的是主循环的概念，后面将详细讨论。

GModule 提供动态加载模块的功能，wdl 未引入动态模块，不予讨论，有兴趣的读者可以参考官方文档。

GThread 顾名思义，提供了多线程的支持。

GIO 提供了较高层的文件系统 API。

2.1.3 GmainLoop

GMainLoop（the main event loop）管理 GLIB 和 GTK+应用程序中所有事件的源（sources）。事件源可以理解为直接导致事件发生的对象。事件可以任意数量，任意源；比如文件描述符（普通文件，管道或者套接字）或者定时器。一般用 g_source_attach()将一个新事件添加到 GMainLoop 中。

为了让多个独立的源可以在不同线程中被处理，每个源与一个 GMainContext 关联。GMainContext 表示主循环中事件源的集合；一个 GMainLoop 有且仅有一个 GMainContext。一个 GMainContext 只能在一个线程中执行，但事件源可以在其他线程中添加或者删除。比如，在 A 线程中可以为 B 线程中的 GMainContext 添加或删除事件源。

每个事件源都有一个相关优先级。默认优先级为 G_PRIOPRITY_DEFAULT，其值为 0。小于 0 的值表示更高的优先级，反之，大于 0 表示低优先级。拥有高优先级源的事件总是比低优先级源相关的事件优先执行。

也可以添加空闲（idle）函数以及相关的优先级。没有更高优先级的事件要执行时就会被执行。

GMainLoop 表示一个主事件循环。使用 g_main_loop_new()创建。添加了初始化的事件源后，调用 g_main_loop_run()。这会不断检查事件源中的事件，然后执行。最后，其中一个事件中调用了 g_main_loop_quit()就会退出主循环，也就是 g_main_loop_run()将返回。

可以通过递归的方式创建主循环。这也是 GTK 应用程序用来显示模态对话框所采用的方式。注意，事件源有个相关的 GMainContext，而该 GMainContext 中所有相关的主循环都会检查和事件源的事件。

GTK+中有一些函数是对 GMainLoop 的封装，比如 gtk_main(),gtk_main_quit()和 gtk_events_pending()。

下面代码是 GMainLoop 的 C 结构体，一个 GMainLoop 只有一个 GMainContext，一个 GMainContext 则可以关联多个事件源。可以看到，GMainLoop 只是 GMainContext 的简单封装，因此，很多时候可以将 GMainLoop，也就是主循环直接看作 GMainContext。

```
struct _GMainLoop{
      GMainContext *context;
      gboolean is_running;
      gint ref_count;
};
```

2.1.4 创建新的事件源类型

GMainLoop 有个不寻常的功能，就是除了可以使用内置的事件源类型外，还可以自定义事件源类型。一个新事件类型可以用来处理 GDK 事件（鼠标点击等）。自定义事件类型一般源于 GSource 结构。自定义事件源类型的结构体内部以 GSource 结构作为第一个元素，其他元素则作为该事件源特有的结构（这种方式和 GObject 的类继承十分相似，不过这里没有采用 GObject）。新建一个事件源对象，调用 g_source_new()，需要指定对事件源具体操作的函数和事件源结构体的大小。

2.1.5 自定义主循环过程

调用 g_main_context_iteration()将完成单步的 GMainContext 迭代。

2.1.6 主循环过程

所谓主循环过程，就是指当我们调用 g_main_loop_run()时，究竟发生了什么。源代码在 glib/gmain.c:3858。

2.2 libcurl

libcurl...

2.3 libtransmission

libtransmission...

3. wdl 的实现

3.1 模块划分

3.2 界面设计

3.3 程序启动

3.4 下载任务

3.5 多线程控制