Carlos Garcia Campos

WebKitGTK+ 2.14

SEPTEMBER 20, 2016

These six months has gone so fast and here we are again excited about the new WebKitGTK+ stable release. This is a release with almost no new API, but with major internal changes that we hope will improve all the applications using WebKitGTK+.

The threaded compositor

This is the most important change introduced in WebKitGTK+ 2.14 and what kept us busy for most of this release cycle. The idea is simple, we still render everything in the web process, but the accelerated compositing (all the OpenGL calls) has been moved to a secondary thread, leaving the main thread free to run all other heavy tasks like layout, JavaScript, etc. The result is a smoother experience in general, since the main thread is no longer busy rendering frames, it can process the JavaScript faster improving the responsiveness significantly. For all the details about the threaded compositor, read Yoon's post here.

So, the idea is indeed simple, but the implementation required a lot of important changes in the whole graphics stack of WebKitGTK+.

- Accelerated compositing always enabled: first of all, with the threaded compositor the accelerated mode is always enabled, so we no longer enter/exit the accelerating compositing mode when visiting pages depending on whether the contents require acceleration or not. This was the first challenge because there were several bugs related to accelerating compositing being always enabled, and even missing features like the web view background colors that didn't work in accelerated mode.
- Coordinated Graphics: it was introduced in WebKit when other ports switched to do the compositing in the UI process. We are still doing the compositing in the web process, but being in a different thread also needs coordination between the main thread and the compositing thread. We switched to use coordinated graphics too, but with some modifications for the threaded compositor case. This is the major change in the graphics stack compared to the previous one.
- Adaptation to the new model: finally we had to adapt to the threaded model, mainly due to the fact that some tasks that were expected to be synchronous before became asyncrhonous, like resizing the web view.

September 2016

M	T	W	T	F	S
			1	2	3
5	6	7	8	9	10
12	13	14	15	16	17
19	20	21	22	23	24
26	27	28	29	30	

« Mar Fe

Tags

cairo cpufreq dbus epiphany evince gnome-applets gnome-panel gtk+ libspectre pdf poppler WebDriver WebKit xps

Categories

Free Software (92)
freedesktop.org (4)
GNOME (88)
WebKit (21)
General (33)
Igalia (30)
Uncategorized (1)

Links

This is a big change that we expect will drastically improve the performance of WebKitGTK+, especially in embedded systems with limited resources, but like all big changes it can also introduce new bugs or issues. Please, file a bug report if you notice any regression in your application. If you have any problem running WebKitGTK+ in your system or with your GPU drivers, please let us know. It's still possible to disable the threaded compositor in two different ways. You can use the environment variable *WEBKIT_DISABLE_COMPOSITING_MODE* at runtime, but this will disable accelerated compositing support, so websites requiring acceleration might not work. To disable the threaded compositor and bring back the previous model you have to recompile WebKitGTK+ with the option *ENABLE_THREADED_COMPOSITOR=OFF*.

Wayland

WebKitGTK+ 2.14 is the first release that we can consider feature complete in Wayland. While previous versions worked in Wayland there were two important features missing that made it quite annoying to use: accelerated compositing and clipboard support.

Accelerated compositing

More and more websites require acceleration to properly work and it's now a requirement of the threaded compositor too. WebKitGTK+ has supported accelerated compositing for a long time, but the implementation was specific to X11. The main challenge is compositing in the web process and sending the results to the UI process to be rendered on the actual screen. In X11 we use an offscreen redirected XComposite window to render in the web process, sending the XPixmap ID to the UI process that renders the window offscreen contents in the web view and uses XDamage extension to track the repaints happening in the XWindow. In Wayland we use a nested compositor in the UI process that implements the Wayland surface interface and a private WebKitGTK+ protocol interface to associate surfaces in the UI process to the web pages in the web process. The web process connects to the nested Wayland compositor and creates a new surface for the web page that is used to render accelerated contents. On every swap buffers operation in the web process, the nested compositor in the UI process is automatically notified through the Wayland surface protocol, and new contents are rendered in the web view. The main difference compared to the X11 model, is that Wayland uses EGL in both the web and UI processes, so what we have in the UI process in the end is not a bitmap but a GL texture that can be used to render the contents to the screen using the GPU directly. We use gdk_cairo_draw_from_gl() when available to do that, falling back to using glReadPixels() and a cairo image surface for older versions of GTK+. This can make a huge difference, especially on embedded devices, so we are considering to use the nested Wayland compositor even on X11 in the future if possible.

Clipboard

The WebKitGTK+ clipboard implementation relies on GTK+, and there's nothing X11 specific in there, however clipboard was read/written directly by

freedesktop.org
GNOME
GNOME Hispano
Igalia
LibreSoft
Linups
Planet GNOME
Planet Igalia
Planeta GNOME Hispano

Archives

September 2017
May 2017
March 2017
February 2017
September 2016
March 2016
September 2015
March 2015
August 2014
July 2014
March 2014
December 2013
September 2013
April 2013
December 2012
August 2012
July 2012
March 2012
February 2012
January 2012
December 2011
November 2011
May 2011
April 2011
March 2011
February 2011
September 2010
July 2010
June 2010
February 2010
September 2009
May 2009
February 2009
January 2009
May 2008
March 2008
December 2007
November 2007
October 2007
September 2007

the web processes. That doesn't work in Wayland, even though we use GtkClipboard, because Wayland only allows clipboard operations between compositor clients, and web processes are not Wayland clients. This required to move the clipboard handling from the web process to the UI process. Clipboard handling is now centralized in the UI process and clipboard contents to be read/written are sent to the different WebKit processes using the internal IPC.

Memory pressure handler

The WebKit memory pressure handler is a monitor that watches the system memory (not only the memory used by the web engine processes) and tries to release memory under low memory conditions. This is quite important feature in embedded devices with memory limitations. This has been supported in WebKitGTK+ for some time, but the implementation is based on cgroups and systemd, that is not available in all systems, and requires user configuration. So, in practice nobody was actually using the memory pressure handler. Watching system memory in Linux is a challenge, mainly because /proc/meminfo is not pollable, so you need manual polling. In WebKit, there's a memory pressure handler on every secondary process (Web, Plugin and Network), so waking up every second to read /proc/meminfo from every web process would not be acceptable. This is not a problem when using cgroups, because the kernel interface provides a way to poll an EventFD to be notified when memory usage is critical.

WebKitGTK+ 2.14 has a new memory monitor, used only when cgroups/systemd is not available or configured, based on polling /proc/meminfo to ensure the memory pressure handler is always available. The monitor lives in the UI process, to ensure there's only one process doing the polling, and uses a dynamic poll interval based on the last system memory usage to read and parse /proc/meminfo in a secondary thread. Once memory usage is critical all the secondary processes are notified using an EventFD. Using EventFD for this monitor too, not only is more efficient than using a pipe or sending an IPC message, but also allows us to keep almost the same implementation in the secondary processes that either monitor the cgroups EventFD or the UI process one.

Other improvements and bug fixes

Like in all other major releases there are a lot of other improvements, features and bug fixes. The most relevant ones in WebKitGTK+ 2.14 are:

- The HTTP disk cache implements speculative revalidation of resources.
- The media backend now supports video orientation.
- Several bugs have been fixed in the media backend to prevent deadlocks when playing HLS videos.
- The amount of file descriptors that are kept open has been drastically reduced.
- Fix the poor performance with the modesetting intel driver and DRI3 enabled.

BOOKMARK THE PERMALINK.

← WebKitGTK+ 2.12

Accelerated compositing in WebKitGTK+ 2.14.4 →

8 COMMENTS

Ernst September 20, 2016 at 11:56 am

Is there a collection of all flags like ENABLE_THREADED_COMPOSITOR somewhere?
I really like about:support in Firefox which show the current status of everything.
And chrome://flags which can say _why_ something is enabled or disabled.

Then my main gripe for actually using Epiphany is missing mousewheel scroll acceleration and I don't like the tab management…

Reply ↓

Ernst September 20, 2016 at 11:57 am

Forgot to add: Kudos to the all the new and advanced features, it's very impressive! 🙂

Reply ↓

Emanuele aina September 20, 2016 at 1:32 pm

Wheeeee! 😀

Thank you for pushing the Wayland AC work forward!

Reply ↓

gapseler September 21, 2016 at 2:48 am

Great release. Thnks for it.

Reply ↓

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name

Email

Website

Save my name, email, and website in this browser for the next time I comment.

CAPTCHA Code *

Post Comment

Powered by WordPress & simpleX.