With all conferences cancelled this year, one of the things I decided to do was trying an online course. After the first week, I was having trouble sleeping at night, with pain at the back of my eyes. I was almost going to give up. I made various adjustments to dramatically reduce lighting. I already had my monitor permanently in the mode for Low Blue Light. I combined this with the GNOME desktop's Night Light mode and configured smart bulbs in my home to run dimmer and redder. After these changes, I feel the problem was almost immediately resolved.

[...]

With record numbers of people being infected, dying or simply losing jobs, the idea of throwing a party may not be the most sensitive thing to do. Nonetheless, if you do want to recreate the feeling of going out to a bar or nightclub, it has never been easier or cheaper to do so with various technologies you can buy online.

Once again, the key theme is lighting. Smart bulbs can be configured for gradual changes throughout the work day. Some bulbs can be configured for more advanced effects coupled with music. One of my lights is a LED panel that can simulate a disco, candlelight, thunderstorm or even a police car to make the night complete. This means it can be anything from a night club in Berlin to a local pub in Ireland.

For sound, seeking a larger diameter speaker sometimes makes a dramatic impact. A pair of oversized vintage speakers from an op-shop may produce better sound than the built-in speakers of most modern laptops, monitors and flat screen TVs.
Once you've conquered light and sound, it is time for taste. An Air Fryer can make chips and there are plenty of recipes suitable for any level of cooking skills. Some of the best models are not available in every store and most of them are a lot cheaper online anyway. While they sound like a jet engine, there is no evidence that Air Fryers have been used in the astronaut diet.

Claudio Saavedra's Changelog - October 2020

In this line of work, we all stumble at least once upon a problem that turns out to be extremely elusive and very tricky to narrow down and solve. If we're lucky, we might have everything at our disposal to diagnose the problem but sometimes that's not the case? and in embedded development it's often not the case. Add to the mix proprietary drivers, lack of debugging symbols, a bug that's very hard to reproduce under a controlled environment, and weeks in partial confinement due to a pandemic and what you have is better described as a very long lucid nightmare. Thankfully, even the worst of nightmares end when morning comes, even if sometimes morning might be several days away. And when the fix to the problem is in an unimaginable place, the story is definitely one worth telling.

[...]

It all started with one of Igalia's customers deploying a WPE WebKit-based browser in their embedded devices. Their CI infrastructure had detected a problem caused when the browser was tasked with creating a new webview (in layman terms, you can imagine that to be the same as opening a new tab in your browser). Occasionally, this view would never load, causing ongoing tests to fail. For some reason, the test failure had a reproducibility of ~75% in the CI environment, but during manual testing it would occur with less than a 1% of probability. For reasons that are beyond the scope of this post, the CI infrastructure was not reachable in a way that would allow to have access to running processes in order to diagnose the problem more easily. So with only logs at hand and less than a 1/100 chances of reproducing the bug myself, I set to debug this problem locally.

[...]

Something that is worth mentioning before we move on is how the WPEBackend-fdo Wayland display integrates with the system. This display is a nested display, with each web view a client, while it is itself a client of the system's Wayland display. This can be a bit confusing if you're not very familiar with how Wayland works, but fortunately there is good documentation about Wayland elsewhere.

The way that the Wayland display in the UI process of a WPEWebKit browser is integrated with the rest of the program, when it uses WPEBackend-fdo, is through the GLib main event loop. Wayland itself has an event loop implementation for servers, but for a GLib-powered application it can be useful to use GLib's and integrate Wayland's event processing with the different stages of the GLib main loop. That is precisely how WPEBackend-fdo is handling its clients' events. As discussed earlier, when a new client is created a pair of connected sockets are created and one end is given to Wayland to control communication with the client.
GSourceFunc functions are used to integrate Wayland with the application main loop. In these functions, we make sure that whenever there are pending messages to be sent to clients, those are sent, and whenever any of the client sockets has pending data to be read, Wayland reads from them, and to dispatch the events that might be necessary in response to the incoming data. And here is where things start getting really strange, because after doing a bit of fprintf()-powered debugging inside the Wayland-GSourceFuncs functions, it became clear that the Wayland events from the clients were never dispatched, because the dispatch() GSourceFunc was not being called, as if there was nothing coming from any Wayland client. But how is that possible, if we already know that the web process client is actually trying to get the Wayland registry?

To move forward, one needs to understand how the GLib main loop works, in particular, with Unix file descriptor sources. A very brief summary of this is that, during an iteration of the main loop, GLib will poll file descriptors to see if there are any interesting events to be reported back to their respective sources, in which case the sources will decide whether to trigger the dispatch() phase. A simple source might decide in its dispatch() method to directly read or write from/to the file descriptor; a Wayland display source (as in our case), will call wl_event_loop_dispatch() to do this for us. However, if the source doesn't find any interesting events, or if the source decides that it doesn't want to handle them, the dispatch() invocation will not happen. More on the GLib main event loop in its API documentation.

**GSoD Weekly Summary 6 [4]**

Since last week, I have been working on the GNOME calculator app and I spent most of my time writing the docs for the different calculator modes.

**Source URL:** [http://www.tuxmachines.org/node/143844](http://www.tuxmachines.org/node/143844)

**Links:**
[1] [http://www.tuxmachines.org/taxonomy/term/146](http://www.tuxmachines.org/taxonomy/term/146)