WAYLAND AND RUST

Headache-oriented code generation

Victor Berger

September 21, 2015

Rust Meetup in Paris

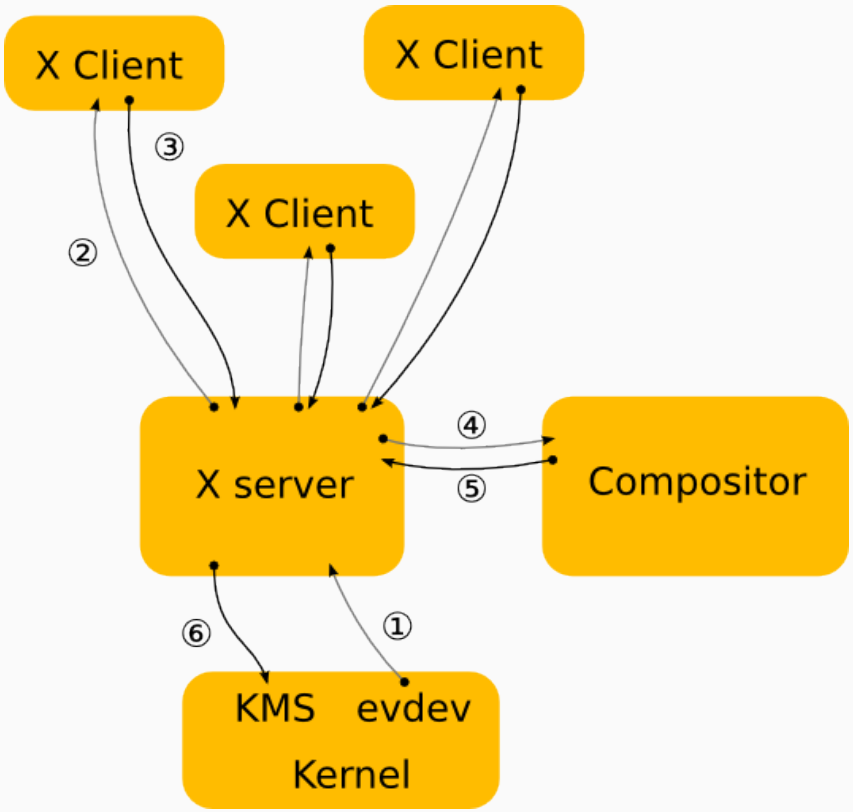
Outline

WHAT IS WAYLAND?

Wayland: the big idea

X.org is bloated, let's do something lighter and more modern.

X11 STRUCTURE

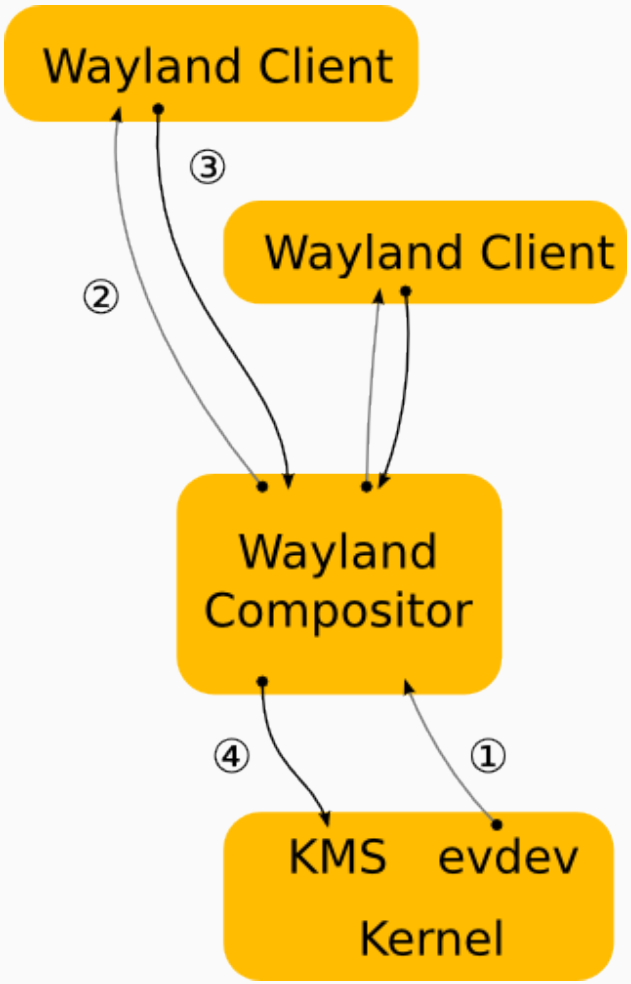


What is the X server for ?

All it does is forwarding events and calls, and is big and slow.

Do we really need it ?

Wayland structure



Wayland is the protocol between the clients and the compositors.

Each DE is potentially a compositor (Gnome-shell, Kwin, ...)

No longer One Big Server [tm] everybody must use.

Wayland structure

In practice: wayland does not offer any drawing primitives.

You just pass to the compositor a buffer containing the pixels to display.

The client handles the drawing itself (OpenGL, cairo, etc...)

Protocol structure

Object-based, with requests and events (bidirectional message bus).

First global object of the connection: `wl_display`.

It provides the second global object: `wl_registry`.

Protocol structure

Then, the registry gives access to the other global objects (`wl_compositor`, `wl_shm`, `wl_seat`, `wl_shell`, ...)

Each of them allows you to create the objects to do your work (`wl_surface`, `wl_buffer`, `wl_keyboard`, `wl_shell_surface`, ...)

How it is classically done

Call functions to do the requests in your objects.

Provide callbacks to handle the events.

CODE GENERATION

The C library, why, how ?

Protocol defined in XML files.

The libraries `libwayland-client.so` handles serialization of it on the socket.

Use this C-library and bind to it ? Or write a 100% Rust library ?

The C library, why, how ?

No choice but use the C-library: mesa links against it and expects its internal structure.

If we want OpenGL support, we need mesa.

We want OpenGL support.

Handling the C library

Very generic library, only does serialization.

Most used function:

```
void wl_proxy_marshal(struct wl_proxy *proxy, uint32_t opcode,...)
```

This function encodes almost all requests, with `void*`s in varargs.

Handling the C library

These crazy functions are not to be directly used.

Goal: use the .xml files to generate a nice and shiny Rust API wrapping this dark magic.

Events handling

Providing callbacks? Do we really want to do this in Rust?

This would be a lifetime nightmare. (It is, I tried).

Lets use iterators, shall we? Much more rusty.

(We'll *just* need to handle an array of `void*` to get the arguments).

Putting things together

A small rust utility that parses the XML file and generate a big `.rs` file wrapping the whole interface.

Plug this file into the minimalist library (yay for `include!(...)` !).

HOW TO USE IT?

How to use it?

Examples on Github (more to come):

`https://github.com/vberger/wayland-client-rs/`

For now, the lib on crates.io is still the previous, hand-written version (including the callback-lifetime nightmare).

The new version does not yet compile on rust stable (but I'm working on it!).

BEHIND THE SCENES

I’m also working with the creator of glutin (1) to integrate it for native wayland support without the need for the user to think about it.

Glutin is the window-handling backend used by servo.

It’s not an easy problem, but if you are motivated, any help would be very welcome !

(1): `https://github.com/tomaka/glutin`

Questions?