X11 and Wayland

A tale of two implementations
X11 and Wayland

A tale of two implementations
What is hikari and what am I trying to achieve?

- window manager / compositor
- started 1.5 years ago
- written from scratch
- stacking / tiling hybrid approach inspired by cwm
- tiling algorithm inspired by herbstluftwm
- keyboard driven, for fast navigation
- modal, inspired by vim
- waste little screen space
- allows to arbitrarily group windows
- minimal dependencies
- energy efficient
- target FreeBSD ❤
- X11 and Wayland implementation
X Window System Architecture vs W
TinyWM is written by Nick Welch <nick@incise.org> in 2005 & 2011.

This software is in the public domain and is provided AS IS, with NO WARRANTY.

```c
#include <X11/Xlib.h>

#define MAX(a, b) ((a) > (b) ? (a) : (b))

int main(void) {
    Display *dpy;
    XWindowAttributes attr;
    XButtonEvent start;
    XEvent ev;

    if (!(dpy = XOpenDisplay(0x0))) return 1;

    XGrabKey(dpy, XKeysymToKeycode(dpy, XStringToKeysym("F1")), Mod1Mask, DefaultRootWindow(dpy), True, GrabModeAsync, GrabModeAsync);
    XGrabButton(dpy, 1, Mod1Mask, DefaultRootWindow(dpy), True, ButtonPressMask|ButtonReleaseMask|PointerMotionMask, GrabModeAsync, GrabModeAsync, None, None);
    XGrabButton(dpy, 3, Mod1Mask, DefaultRootWindow(dpy), True, ButtonPressMask|ButtonReleaseMask|PointerMotionMask, GrabModeAsync, GrabModeAsync, None, None);

    start.subwindow = None;

    for (;;) {
        XNextEvent(dpy, &ev);
        if (ev.type == KeyPress && ev.xkey.subwindow != None) XRaiseWindow(dpy, ev.xkey.subwindow);
        else if (ev.type == ButtonPress && ev.xbutton.subwindow != None) {
            XGetWindowAttributes(dpy, ev.xbutton.subwindow, &attr);
            start = ev.xbutton;
        } else if (ev.type == MotionNotify && start.subwindow != None) {
            int xdiff = ev.xbutton.x_root - start.x_root;
            int ydiff = ev.xbutton.y_root - start.y_root;
            XMoveResizeWindow(dpy, start.subwindow, attr.x + (start.button==1? xdiff : 0),
                              attr.y + (start.button==1? ydiff : 0),
                              MAX(1, attr.width + (start.button==3? xdiff : 0)),
                              MAX(1, attr.height + (start.button==3? ydiff : 0)));
        } else if (ev.type == ButtonRelease) start.subwindow = None;
    }
}
```
Talking to the X Server

Xlib

XCB

Writing request
Stalled, waiting for data
Reading reply

---

#include <X11/Xlib.h>

#define MAX(a, b) ((a) > (b) ? (a) : (b))

int main(void)
{
  Display * dpy;
  XWindowAttributes attr;
  XButtonEvent start;
  XEvent ev;

  if(!dpy = XOpenDisplay(0x0)) return 1;

  XGrabKey(dpy, XKeysymToKeycode(dpy, XStringToKeysym("F1")), Mod1Mask, DefaultRootWindow(dpy), True, GrabModeAsync, GrabModeAsync);
  XGrabButton(dpy, 1, Mod1Mask, DefaultRootWindow(dpy), True, ButtonPressMask|ButtonReleaseMask|PointerMotionMask, GrabModeAsync, GrabModeAsync, None, None);
  XGrabButton(dpy, 3, Mod1Mask, DefaultRootWindow(dpy), True, ButtonPressMask|ButtonReleaseMask|PointerMotionMask, GrabModeAsync, GrabModeAsync, None, None);

  start.subwindow = None;
  for(;;){
    XNextEvent(dpy, &ev);
    if(ev.type == KeyPress && ev.xkey.subwindow != None)
      XRaiseWindow(dpy, ev.xkey.subwindow);
    else if(ev.type == ButtonPress && ev.xbutton.subwindow != None) {
      XGetWindowAttributes(dpy, ev.xbutton.subwindow, &attr);
      start = ev.xbutton;
    } else if(ev.type == MotionNotify && start.subwindow != None) {
      int xdiff = ev.xbutton.x_root - start.x_root;
      int ydiff = ev.xbutton.y_root - start.y_root;
      XMoveResizeWindow(dpy, start.subwindow, attr.x + (start.button==1 ? xdiff : 0), attr.y + (start.button==1 ? ydiff : 0), MAX(1, attr.width + (start.button==3 ? xdiff : 0)), MAX(1, attr.height + (start.button==3 ? ydiff : 0)));
    } else if(ev.type == ButtonRelease)
      start.subwindow = None;
  }
}
Talking to the X Server

Xlib

W-----RW-----RW-----RW-----R

XCB

WWWW--RRRR

- Writing request
- Stalled, waiting for data
- Reading reply
Window ordering

Window 2

Window 1
I can haz keyboardz plz?
// taken from awesome keygrabber.c
static bool keygrabber_grab()
{
    xcb_grab_keyboard_reply_t *xgb;
    for (int i = 1000; i; i--) {
        if ((xgb = xcb_grab_keyboard_reply(globalconf.connection,
                                            xcb_grab_keyboard(globalconf.connection,
                                            true,
                                            globalconf.screen->root,
                                            XCB_CURRENT_TIME, XCB_GRAB_MODE_ASYNC,
                                            XCB_GRAB_MODE_ASYNC),
                                            NULL)) {
            p_delete(&xgb);
            return true;
        }
        usleep(1000);
    }
    return false;
}
I can haz keyboardz plz?

// taken from awesome keygrabber.c
static bool keygrabber_grab(void)
{
    xcb_grab_keyboard_reply_t *xgb;

    for(int i = 1000; i; i--)
    {
        if((xgb = xcb_grab_keyboard_reply(globalconf.connection, xcb_grab_keyboard(globalconf.connection, true, globalconf.screen->root, XCB_CURRENT_TIME, XCB_GRAB_MODE_ASYNC, XCB_GRAB_MODE_ASYNC), NULL)))
        {
            p_delete(&xgb);
            return true;
        }
        usleep(1000);
    }

    return false;
}
Conclusion

- very easy to get something up and running
- graphical user interfaces have evolved
- "gazillions" of X extensions (legacy demands it)
- global name space (bad security implications)
- window manager is just a client
- duplicating functionality in the window manager
- screen artifacts (gets a bit better with COMPOSITE)
Wayland Architecture

Wayland Client

Wayland Compositor

KMS evdev

Kernel
### Every frame is perfect!

<table>
<thead>
<tr>
<th></th>
<th>Client A</th>
<th>Client B</th>
<th>Compositor</th>
</tr>
</thead>
<tbody>
<tr>
<td>draw</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>draw</td>
<td></td>
<td></td>
<td>draw</td>
</tr>
<tr>
<td>draw</td>
<td></td>
<td></td>
<td>draw</td>
</tr>
<tr>
<td>draw</td>
<td></td>
<td></td>
<td>no need to draw</td>
</tr>
<tr>
<td>draw</td>
<td></td>
<td></td>
<td>draw</td>
</tr>
</tbody>
</table>

[1] [https://emersion.fr/blog/2019/intro-to-damage](https://emersion.fr/blog/2019/intro-to-damage)
Pluggable, composable, unopinionated modules for Wayland compositor; or about 50,000 lines of code you'd have to write anyway.


- written in C
- used by sway [3] https://swaywm.org/
- 0.1 release Oct 21, 2018
- provides a common ground for many compositors

Interesting compositors based on wlroots

- tinywl ~1KLOC (shipped with wlroots)
Conclusion

- It's harder to get something up and running slightly more code to have the same functionality I had with X11.
- Fewer processes involved (no duplicated functionality).
- UI Isolation.
- Way less complexity.
- Direct control over devices.
- Control over frames (no flickering, no tearing, no flashes).
- Client side decorations.
- More responsibility on the compositor.
- Large toolkit support.
- Great opportunity for Open Source systems to catch up.

Toolkits

- GTK: `GDK_BACKEND=wayland`
- Qt: `QT_QPA_PLATFORM=wayland-egl`
- Clutter: `CLUTTER_BACKEND=wayland`
- SDL: `SDL_VIDEODRIVER=wayland`

Applications

- Firefox / Thunderbird: `MOZ_ENABLE_WAYLAND`
- mpv
- `wl-clipboard` (makes my `neovim` happy)

Running X Applications on Wayland

- `Xwayland` (needs compositor support)
Conclusion

- it's harder to get something up and running
- slightly more code to have the same functionality I had with X11
- fewer processes involved (no duplicated functionality)
- UI isolation
- way less complexity
- direct control over devices
- control over frames (no flickering, no tearing, no flashes)
- client side decorations
- more responsibility on the compositor
- large toolkit support
- great opportunity for Open Source systems to
The compositor part of Way Cooler is now written in C. The client portion (i.e. the side that implements the AwesomeWM functionality) is still written in Rust. Ultimately, wlroots-rs was too difficult to write. The mental overhead of attempting to wrap complicated C libraries with Rust is too demanding. This complexity often leads to a RiiR mindset, which I am strongly against. So, the compositor is now written in C.
The compositor part of Way Cooler is now written in C, portion (i.e. the side that implements the AwesomeWM functionality) is still written in Rust.

Ultimately, wlroots-rs was too difficult to write. The overhead of attempting to wrap complicated C libraries with Rust is too demanding. This complexity often leads to a RiiR mindset, which I am strongly against. So, the compositor is now written in C.

Thank you!

Contact Mastodon: chaos.social/@raichoo
Matrix: @raichoo:acmelabs.space
Hikari Matrix Chat: #hikari:acmelabs.space

ASAN

```
clang -fsanitize=address
```

```
0x50200000ef3d is located 8 bytes to the right of 13-byte region [b5555555]|[b5555555]|0x50200001|
allocated by thread 10 here:
#0 0x10810ebf0 in wrap_malloc (libclang_rt.asan.osx_dynamic.dylib+d777)
#1 0x10813bc85 in main clang-asan.c:6
#2 0x7f66c46254 in start (libdyld.dylib+0x5254)

SUMMARY: AddressSanitizer: heap-buffer-overflow clang-asan.c:10 in main
Shadow bytes around the buggy address:

 0x1c000001d90: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c000001d98: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c000001da0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c000001d98: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c000001da0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c000001d90: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c000001d98: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c000001da0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c000001d90: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c000001d98: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c00001d90: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c00001d98: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x1c00001da0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
```
Thank you!

Contact

- Mastodon: chaos.social/@raichoo
- Matrix: @raichoo:acmelabs.space
- Hikari Matrix Chat: #hikari:acmelabs.space