WAYCRATE

SWHKD: PROVIDE LIBINPUT AS AN ALTERNATIVE INPUT BACKEND

GSoC 2025

Mohammad Aadil Shabier
:aadilshabier

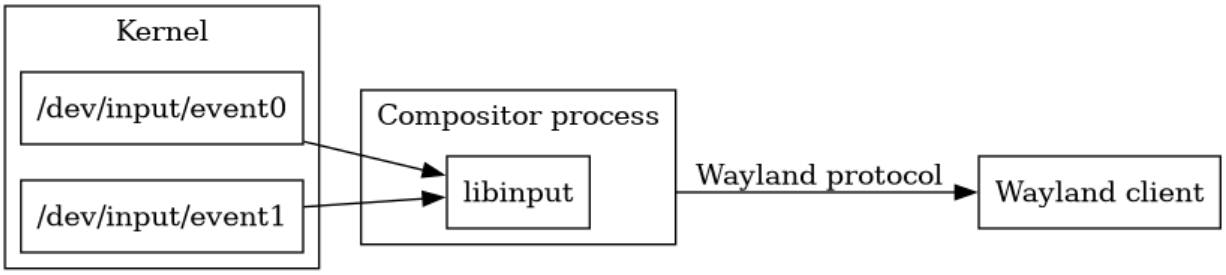
Contact Information

- **Email:** aadilshabier1@gmail.com
- **Github:** aadilshabier
- **Linkedin:** Mohammad Aadil Shabier
- **Discord:** tps787

Introduction

The project I'm planning to contribute to is related to a highlighted issue from the ideas page(#191). It seeks to replace evdev with a libinput backend.

swhkd currently uses *udev* and *libevdev* for input handling. *Udev* is a device manager which is responsible for detecting when devices are added and removed, these devices are then "grabbed" so that all events from these devices pass through *swhkd*. *Evdev* is a generic input interface which passes events generated from the kernel to the userspace, it is used to get all events coming from these devices as a stream which it then processes. We then check for any keybinds or shortcuts, any other events are simulated using *uinput*, a kernel module which is used to emulate input devices from the user space using *libevdev*.
Libinput is a user space input library which processes events from evdev devices. It offers excellent support for commonly used input devices like mice, keyboards, touchpads, etc. In short, it is built over *evdev* and handles filtering, debouncing, acceleration, gestures, etc to provide a better user experience.



The goal of this project is to provide an option for the user to use *libinput* as the input backend.

Related Experience

You can find my resume here, and some other experiences on my Linkedin. I primarily work in **Systems Engineering** and **Machine Learning** in C, C++, Python, Rust, etc.

I love to program, primarily in C++ and Python. I participated in GSoC 2022 under **Inkscape**, where I worked in C++, on modernizing memory management and worked towards removing the garbage collector.

I've contributed to scalpel, a packet dissector and sculptor written in Rust.

At the time of writing, this I have made a few PRs to *swhkd*.

Why Waycrate?

I came across Rust a few years ago, and was pretty amazed by the ease of programming in it compared to some other languages. The packet management seemed as easy as in Python, while having all the raw performance possible in C, with a few cool bells and whistles. Working on the Rust libraries I mentioned above made me realize how much I enjoyed programming in Rust. So I hopefully get to do GSoC 2025 at Waycrate, and get to work on something used by a lot of people.

I really like the vision of the Waycrate team, and really enjoyed the community I came across on Discord. I think the maintainers have a lot going for them as maintaining such a project is probably by no means easy.

I've been using i3 since 2021 and eventually plan to switch to sway once I have enough time to rice my setup again xD. I think working on a slightly newer project which I will eventually go on to use will be very exciting.

Proposal

Motivation

Currently, we capture input from only the keyboard, mouse, and similar devices (currently defined as devices which support the Enter key) unless specified.
The goal of *libinput* is to provide **better support for common devices** that interact with a desktop environment, this also means that it does not support devices like joysticks because:

a. It does not have a clear interaction with the desktop environment
b. Any abstraction that libinput would provide would introduce complexity and processing delays for no clear benefit.

Libinput handles some **device specific quirks**, and also has code to handle common device malfunctions like debouncing, this could help avoid some pitfalls faced when handling certain types of inputs directly with *evdev*, and consecutively solve some device related issues, e.g: #230, #244, #209, #211, #215, etc.
Libinput also only supports a subset of the switches that *libevdev* supports, so any user that requires these extra switches might find it better to continue using *libevdev*. This project would allow testing the feasibility of this alternative backend, which gives the user more fine grained control over the input backend to better suit their uses.
Libinput does not handle all device types. E.g: joysticks. We may need to handle these separately using some sort of *evdev* passthrough.
Grabbing the devices might also interfere with gestures, so we have to give the user power to ignore any devices if they want fine grained control over their experience.
I also plan to add functionality to emulate device events for devices which we currently do not have, e.g: middle mouse click emulation.

Description

This project will be a medium project lasting for **175 hours**. The work done in this project will be done in a separate branch/repository created for the

express purpose. The project will need consistent input and testing by members of the community as it involves changing elements of the user experience, and removing functionality which may be useful to them.

Phase 1

The community bonding period lasts for around a month, this will be crucial to get to know the Waycrate community, and read existing documentation to plan out all the changes before coding actually starts.

The project would provide *libinput* as an optional build dependency which can be added using Cargo “features”. We would initially develop it as a separate project called *swhkd_libinput*, and we will try to keep parity with the current version in terms of behaviour and command line flags.

Phase 2.1

The coding period starts on June 1. We will start from scratch and try to replicate some existing functionality of *swhkd* using the libinput backend. At this stage, we will not plan changes in any of the other modules in swhkd (config, perms, tests, uinput) but rather work out the specifics of the library and implement the major functionality of swhkd:

1. Adding and removing devices using udev.
2. User arguments to add devices by name.

Phase 2.2

The project will continue using *uinput* to generate virtual device events, this would be similar to the evdev implementation. This is done by implementing a translation layer which convert a libinput event to an *evdev* event to be able to simulate it with *uinput*.
We also implement a feature which allows us to simulating device events for devices and buttons which we currently do not have. This can be used to emulate missing keys, such as middle mouse clicks, pause, play, mute buttons, etc.

Phase 2.3

The current implementation uses an asynchronous API to reduce CPU usage, we use the *AsyncFd* struct in tokio, which provides a wrapper around the libinput Rust interface to make it async.
We then implement the cooldown timer as in the older version.

Phase 2.4

As we discussed above, libinput does not handle non-common devices, although this is the majority of devices, and will work without problems for the vast majority of the people, we can experiment handling events for these devices by using an *evdev* fallthrough.
It is not currently clear whether this will work as expected, but it would be a nice addition if it worked.

Phase 2.5

To provide libinput as a drop-in replacement, we would have to modularize and clean up the code in *swhkd_libinput*. This also allows us to make a minimal amount of changes to replace it with other userspace input libraries in the future. This also enhances readability and maintainability which is important for a project of this scale used by a large number of people.

Phase 3

This phase will prepare for the end of the Summer of Code period by reviewing all the changes that were made, documenting them, writing unit tests and integration tests, and completing the GSoC blog. All the changes that have been made will be ironed out in preparation for the final evaluation.

Timeline

1. Read and Research	1.1 Community bonding period	May 8 - May 17 *1.5 weeks*

	1.2. Plan and document core differences that could arise from changes to the input backend. DIscuss with the community and understand the tradeoffs to make choices.	May 18 - May 31 *2 weeks*
2. Start implementation and plan integration into existing *swhkd*	2.1. Start from scratch and try to replicate some existing functionality of *swhkd* using the libinput backend.	June 1 - June 14 *2 weeks*
	2.2. Experiment with ways to replicate existing usage, such as emitting events, grabbing devices using the *libinput* backend, etc.	June 15 - July 7 *3 weeks*
	2.3. Asynchronize the *libinput* API for better performance and CPU usage and to work well with the current implementation.	July 8 - July 14 *1 week*
	MIDTERM EVALUATION	
	2.4. Implement input handling for joysticks and other devices not handles by libinput	July 15 - July 28 *2 weeks*
	2.5. Modularize code to make it a drop-in replacement	July 29 - August 11 *2 weeks*
3. Test and Document	Test the functionality, write unit tests, document all changes, iron out all changes. Bring everything to completion.	August 12 - September 1 *2.5 weeks*
FINAL EVALUATION		

Availability

After my college ends in the first week of May, I'll have most of the day to dedicate to the project.

References

1. Input subsystem (link)
2. Kernel input programming documentation (link)
3. Thread with information about grabbing *libinput* devices. (link)
4. Work that was done previously (link)
5. How *libinput-record* (link) and *libinput-replay* (link) emits events.