Project Proposal

I propose to work on the creation of a DMA-BUF backend for waycrate's libwayshot screen capture library. Presently libwayshot's method of screen-capture requires copying from graphics memory to system memory via CPU using the shared memory methods of the wlr-screencopy protocol. This method has too much overhead for uses like high resolution streaming: any further transforms and processing of the image will either be performed on CPU (inefficient) or require another copy back onto graphics memory. There is a better way to do this.

The DMA-BUF (short for Direct Memory Access Buffer) is a Linux kernel framework that allows for efficiently sharing buffers of memory between multiple graphics devices or subsystems. Using the DMA-BUF export mechanisms in wlroots we can completely avoid downloads and uploads between graphics memory and the CPU, thus allowing for increased efficiency and performance. This will be especially noticeable on higher resolution screen-captures where individual frame buffers become too large to be efficiently dealt with on a CPU compared to how they can be handled on the GPU.

The Deliverables for this project are: Complete implementation of a high-performance DMA-BUF based screen capture backend in libwayshot, Documentation for using it.

Related Experience

- Built midirouter which (with the MIDI audio protocol) has functionality roughly analogous to what xdg-portal-luminous has for the wayland ecosystem - move multimedia streams from one application to another.
- Built a toy raytracer in Rust in order to gain an understanding of the process of turning abstract mathematical objects into pixels in a buffer: https://github.com/CheerfulPianissimo/ray_path
- Have an open documentation PR on the sway wayland compositor.
- Minor experience in open source contributions, mostly in small bug-fixes and patches.
- Experience in porting postmarketOS to a new target device to the point of having the fb-dev display driver+xfce4 working.

Why GSoC with Waycrate?

There are several reasons why I think waycrate will be a good fit for me to work with in GSoC 2024:

- I've been daily-driving wayshot for a while and am personally incentivised to improve its functionality.
- I have plans to use libwayshot for future projects that I think will serve unmet community needs. To be more specific, I wish to build a chromecast client for wlroots based compositors. Having performant screen capture is essential for this.
- I've participated in the community and find the members quite amicable and proactive in preparing for the program.
- I wish to have more sustained experience with open source contributions than mere one-off patches and bug-fixes and believe the structured format and constraints of GSoC+waycrate will provide just the right environment to facilitate this.

The Proposal

Here I will detail my proposal for GSoC 2024 under waycrate:

Motivation

libwayshot is a Rust crate that implements screen capture functionality for wlroots based compositors. It's also associated with wayshot - a cli app based on libwayshot that provides an executable for capturing wlroots outputs or parts of them.

Presently libwayshot uses shared memory (in the form of wl_shm buffers) to transfer screen capture data from the compositor via the wlr-screencopy protocol.

However, most wlroots compositors are hardware accelerated and will make use of the VRAM to perform all of their rendering. Transferring graphics data via memfd_create necessitates that the compositor move the required data from the VRAM via the CPU, adding a degree of overhead.

Now, some applications may be just fine with this because they already have no option but to download data from the VRAM in order to encode and store it on the filesystem or send it over the network or other such CPU-centric processes. This is the case for one-off screencapture applications like wayshot.

However, there is a class of applications which need to perform further GPU accelerated operations on this graphics data. Consider:

- Screen recorders that use hardware acceleration for video encoding.
- Desktop portals like xdg-portal-luminous that route screen content to other apps.
- Screen mirroring apps that apply various transformations on screen content before returning it to the compositor to display.

In all these cases the screen content needs to be uploaded back into the GPU. In these cases using shared memory simply adds unnecessary downloads and uploads with respect to the GPU. wlroots

provides the capability to avoid this inefficiency by providing methods to keep the captured content on VRAM without any copies. My motivation here is to propose a way for libwayshot to support these methods and widen its applicability to a larger class of projects that involve screen capture.

Detailed Description

First, we need some context around how wlroots-based wayland compositors handle screen capture. There are mainly two protocols that enable this:

1. wlr-screencopy-unstable-v1
2. wlr-export-dmabuf-unstable-v1

Libwayshot in its present state uses only wlr-screencopy. The major difference between these two is in the type of buffers each of them deal with:

- In wlr-screencopy the compositor will notify the client of upto two different types of available buffer types: wl_shm shared memory buffers and linux dmabuf based DMA-BUF buffers. The client must create one of these and pass it to the compositor for it to copy screen content into.
    - libwayshot presently uses only the wl_shm mode of wlr-screencopy.
    - wlr-screencopy is capable of copy on damage functionality: the compositor signals a filled buffer only when the captured region has been damaged.
    - wlr-screencopy can choose to capture certain sub-regions of outputs.
    - Buffers here must be created by the client: wl_shm buffers can be created by memfd_create and dmabuf buffers can be created by the linux_dmabuf wayland protocol.
        - A copy is unavoidable here. However, VRAM->CPU copies can still be avoided in dmabuf mode as dmabuf buffer copies occur entirely on the GPU. These are relatively efficient depending on the hardware and drivers.

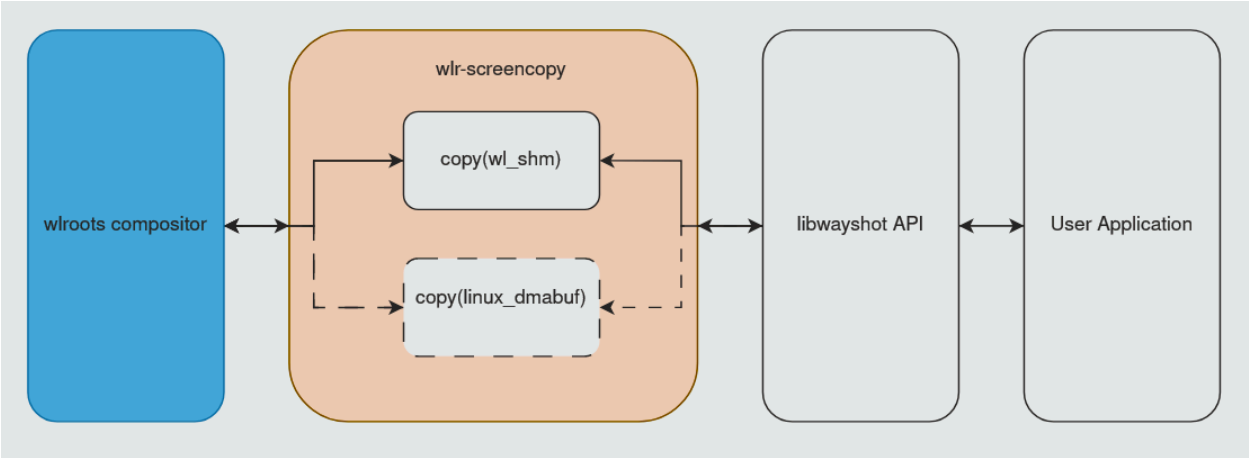
- wlr-export-dmabuf deals solely with dmabuf buffers. In this protocol the compositor passes off dmabuf handles to the client for each frame. No copying will be involved here.
    - The client can choose which output to capture but it cannot select regions under this output.
    - No copy with damage is available.
    - The client need not create dmabuf buffers of its own.

Overall the tradeoffs between these approaches are as follows:

- wlr-screencopy needs to perform dmabuf copies but these are pretty efficient and any performance losses here compared to wlr-export-dmabuf here should be somewhat offset by the availability of copy with damage requiring less data to be processed overall. Ultimately benchmarking will be required to understand the performance gaps.
- wlr-screencopy is more complex to implement than wlr-export-dmabuf but this is offset by the fact that most of this complexity is already present in libwayshot while wlr-export-dmabuf will have to be implemented from scratch.
    - The overall trade off appears to be between the flexibility and universal availability of wlr-screencopy versus the simplicity and (arguably) performance of wlr-export-dmabuf.

After consideration of the above tradeoffs and discussion with organization contributors, I have decided to implement the wlr-screencopy version of the dmabuf backend for this project. Let’s discuss this in detail.

Here's a diagram containing high level overview of what I propose to initially build. The components in dotted lines will be implemented as part of GSoC.



The main steps in implementing this are:

1. dmabuf creation and configuration (using the linux-dmabuf protocol)
2. dmabuf backed wl-buffer creation
3. Passing to compositor for copy
4. Returning the resulting buffer to the client

Let's have a high-level overview for each of these steps:

DMA-BUF Backed wl-buffer Creation

Creating a DMA-BUF is a somewhat more involved process than creating shm buffers. While the latter can be created using a few system calls, the general mechanism for creating DMA-BUFs in wayland is by using the stable linux-dmabuf protocol.

The details of these protocol are somewhat beyond the scope of this proposal but the process has two steps:

1. Obtain the ZwpLinuxDmabufV1 interface from the compositor.
    a. The wayland global registry can be queried to do this.
2. Use ZwpLinuxDmabufV1 to create a ZwpLinuxBufferParamsV1 interface object.
    a. Relevant request: create_params
3. Use the ZwpLinuxBufferParamsV1 interface to build a dmabuf backed wl-buffer.
    a. Relevant request: create_immed

Passing the wl-buffer to compositor for copy

Once the requisite wl-buffer has been created it needs to be passed to the compositor to be filled in. The steps for this are:

1. Check if the compositor actually has dmabuf support. This is done by checking for a linux_dmabuf event from the ZwlrScreencopyFrameV1 interface. This event will have all the information needed to configure the dmabuf later on.
2. Create the dmabuf backed wl-buffer using the linux-dmabuf protocol as discussed above.
3. Pass the dmabuf backed wl-buffer to the compositor to be filled in. libwayshot already does this with the copy request on the ZwlrScreencopyFrameV1 interface. The process is the same for dmabuf backed wl-buffers.

Returning the resulting buffer to the client

Once the copy is complete libwayshot has to pass the resultant dmabuf backed wl-buffer back to the caller in an appropriate form. Further processing and transformations if any are required can be performed here. An ergonomic and useful API has to be designed and the right data types chosen.

Stretch Goals

If time constraints permit I have a few more ideas relating to the main proposal that I would like to implement, in order of their priority:

1. Switching xdg-portal-luminous from screencopy+wl-shm to screencopy+DMA-BUF

   xdg-portal-luminous is waycrate's desktop portal project that provides screenshotting and screencasting capabilities via libwayshot. Presently the screencasting thread here uses libwayshot's **capture_output_frame_shm_fd** to perform screen captures using shared memory backed wl-buffers. This involves a high overhead VRAM download which may not be useful for many applications using the portal. Once dmabuf support is built into libwayshot, it should be possible to modify xdg-portal-luminous to use it to provide improved screen capture performance.

2. Implement a benchmarking suite
   As the old adage goes: "You can't improve what you don't measure." If libwayshot is to aim for high performance applications, we need to be capable of quantifying its performance. Regressions in performance may also be detected easily with such a suite. Plus, it makes goal 4 more feasible.

3. Implement screen recording in wayshot CLI
   The tracking issue for this is here. Discuss with mentors if this is to be attempted within wayshot CLI or as a separate app. Makes more sense if integrated with goal 5 below.

4. Implement and benchmark screen capture with wlr-export-dmabuf
   I discussed the reasons to prioritize wlr-screencopy over wlr-export-dmabuf above. If time permits I can also try building a wlr-export-dmabuf based backend for libwayshot. Benchmarking can be used for finding out how much of a useful route this is over wlr-screencopy. There may be certain cases where the lack of copy on damage doesn’t affect performance, like high resolution, high framerate game streaming. If such applications are found the additional backend may be integrated into libwayshot.

5. Investigate wlr-export-dmabuf for screen recording
   There is an existing project wl-screenrec that uses wlr-screencopy+dmabufs along with hardware video encoding via ffmpeg to create highly performant screen recording applications. Further performance gains could be realized by using wlr-export-dmabuf as the screen capture protocol via libwayshot if goal 3 is implemented. I asked the author of the project about it and they seemed positive: Issue.

Project Timeline Plan

Here's my tentative timeline for GSoC 2024. It is not set in stone and can be changed when the landscape for the project becomes more clear. I have taken a conservative approach with this plan, leaving the two weeks at the end as an emergency buffer. In this spirit, the stretch goals have also not been taken into account.

Period	**Activity**
Community Bonding Period **May 1 to May 26**	
Week 1 **May 1 to May 5**	● Get to know my mentors and fellow waycrate contributors ● Hash out expectations, development conventions and schedules for meetings ● Read up on wayland documentation
Week 2 **May 6 to May 12**	● Read up on and understand the wayland-rs library ● Understand the requisite protocols and their history ● Understand and document overall libwayshot architecture
Week 3 **May 13 to May 19**	● Create tracking issues for the idea implementation ● Improve documentation for libwayshot ● Try building a few small programs using the libwayshot API
Week 4 **May 20 to May 26**	● Focus on end-semester exams ● Fill in more details in the proposal with all research performed till date
Goals For Period: ● Gain an user perspective on the libwayshot API ● Become a part of the waycrate community ● Improve understanding of the codebase ○ Try improving documentation of the codebase in this	

<table>
<tr><td colspan="2">process
● Get used to balancing college coursework and GSoC</td></tr>
<tr><td colspan="2">Official Coding Phase Begins</td></tr>
<tr><td>Week 1
May 27 to June 2</td><td rowspan="2">● Focus on end-semester exams</td></tr>
<tr><td>Week 2
June 3 to June 9</td></tr>
<tr><td colspan="2">Implementation Phase 1 - Tackle linux-dmabuf-v1!
June 10 to June 23</td></tr>
<tr><td>Week 3
June 10 to June 16</td><td>● Understand the linux-dmabuf-v1 protocol
● Understand how wayland-rs can be used to implement it
● Build and test a MVP using it that creates a dmabuf on the GPU</td></tr>
<tr><td>Week 4
June 17 to June 23</td><td>● Integrate the protocol implementation into libwayshot
● Figure out the API design and datatypes for passing the dmabuf to the user
● Document everything done in phase 1</td></tr>
<tr><td colspan="2">Goals For Phase 1:
● Gain familiarity with wayland-rs and other libwayshot deps
● Gain familiarity working on and modifying the libwayshot codebase
● Implement dmabuf creation in libwayshot
Deliverables For Phase 1:
● dmabuf MVP in libwayshot</td></tr>
<tr><td colspan="2">Implementation Phase 2 - Get DMA-BUF backed wl-buffers
July 1 to July 21</td></tr>
<tr><td>Week 5
July 1 to July 7</td><td>● Use linux-dmabuf protocol to create a wl-buffer backed by a dmabuf
● Implement a way to test that the dmabuf setup is actually working</td></tr>
<tr><td>Week 6</td><td>● Focus on end-semester Lab exams</td></tr>
</table>

July 8 to July 14	● GSoC Midterm Evaluations
Week 7 **July 15 to July 21**	● Use screencopy protocol to detect presence of dmabuf capabilities ● Use the information from the linux_dmabuf event to configure the dmabuf created earlier ● Document everything in phase 2
Goals For Phase 1: ● Integrate linux-dmabuf and wlr-screencopy protocols ● Get a basic working example for dmabuf screenshotting **Deliverables For Phase 1:** ● A screenshotting MVP in libwayshot using dmabuf	
Implementation Phase 3 - Pull screen captures into the wl-buffers **July 22 to August 11**	
Week 8 **July 22 to July 28**	● Create the user facing dmabuf backed screen capture API in libwayshot ● Configure it to setup a dmabuf backed wl-buffer and use it in copy or copy_with_damage
Week 9 **July 29 to August 4**	● Test and ensure that the dmabuf backend works with other capabilities of libwayshot ● Implement any integrations required for region selection, output selection, etc
Week 10 **August 5 to August 11**	● Integrate dmabuf backend into wayshot CLI application ● Test the completed app on as many permutations and devices as possible
Goals For Phase 3: ● Finalize and implement user facing API in libwayshot ● Bring out this functionality in wayshot CLI app if such a thing is beneficial ● Test this functionality	

<table>
<tr><td colspan="2">Deliverables For Phase 3:
● Complete implementation of DMA-BUF based screen capture in libwayshot
● Documentation for using this backend</td></tr>
<tr><td>Week 10
August 12 to August 18</td><td rowspan="2">● Buffer period for debugging, ironing out any remaining problems and pursuing stretch goals.
● Document everything done in Phase 3
● Document the GSoC experience, prepare a demo demonstrating improvements if possible
● Perform any further required activities discovered during previous phases</td></tr>
<tr><td>Week 11
August 19 to August 26</td></tr>
<tr><td colspan="2">GSoC Contributor Evaluations</td></tr>
</table>