RFQ about Common Device I/F for both virt and non-virt AGL

This document constitutes request for quotation (RFQ) for Common Device I/F based on VirtIO in virt or non-virt AGL PoC from Automotive Grade Linux Virtualization Expert Group (hereinafter referred to as “Virt-EG”)

It defines requirements for two approaches to achieve common device I/F:

1. Use Linux kernel API and common Linux libraries (existing upper I/F used by VirtIO) as a common device I/F.
2. Use VIRTIO front-end & back-end framework as a common device I/F.

1. Definitions

HAL - Hardware Abstraction Layer, software layer abstracting hardware from middleware and applications. In scope of this document terms HAL and “common device I/F” are used interchangeably and have the same meaning.

PoC - Proof of concept, realization of certain idea suitable to demonstrate its feasibility, but not necessarily complete.

VIRTIO - a standardized interface which allows virtual machines access to "virtual" devices.

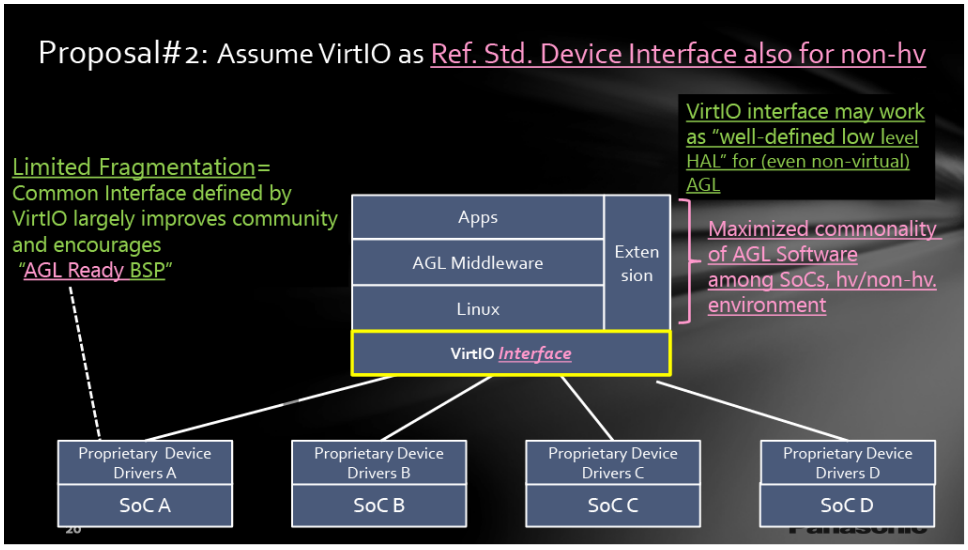
2. Background

2.1. Discussion Background

The topic of common device I/F has been widely discussed already.

2.1.1. Device Virtualization Architecture for Automotive, Nov. 2020, Masashige Mizuyama, CTO, Automotive Company, Panasonic Corporation

https://automotivelinux.org/wp-content/uploads/sites/4/2020/11/AGL_Webinar20201113_mizuyama.pdf, slide 20



2.1.2. SPEC-3855 ("Adapt Product Readiness HAL to VirtIO I/F to separate HAL implementation from device driver implementation"), created 22/Mar/21

https://jira.automotivelinux.org/browse/SPEC-3855

■Goal

Adapt IVI PR HAL to VirtIO I/F

■Benefits

Use VirtIO driver I/F as lower I/F makes userspace Implementation of IVI PR independent from lower level device driver implementation. (increase portability of IVI-PR across different SoC platforms, Native/Virtual Environment)

■Action Item

①Prioritize devices necessary to AGL and define common I/F for AGL (in cooperation with IC-EG, Virt-EG and SAT) --> cross-EG activities to be done in SAT

②Investigate current IVI PR HAL implementation "

③Design & implement to adapt current HAL implementation to VirtIO I/F (defined in SAT)

④Merge to IVI-profile

■Member

Welcome anyone to join.

2.1.3. SPEC-3867 ("[VIRT-003] Define common device I/F for both virtual AGL and non-virtual AGL (native AGL UCB) and implement physical device driver adapted to the common device I/F in non-virt environment"), created 31/Mar/21

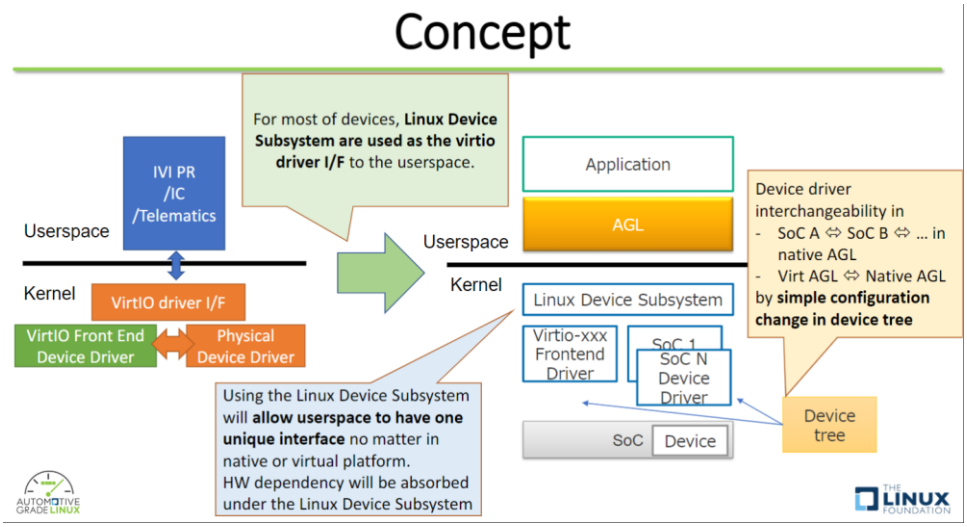
https://jira.automotivelinux.org/browse/SPEC-3867

Step 1: Based on VIRT-002, define common device interface (take advantage of VirtIO driver I/F) for both native and virtual AGL. Cooperation in SAT with IVI-EG, IC-EG and other EGs are needed.
Step 2: Implement reference physical device driver complied with VirtIO I/F defined in step 1 to extend VirtIO use to native environment (as a common device interface)
(see also IVI-008 and SAT-018)

VIRT-002 is "Define necessary devices for virtual AGL and implement respective VirtIO drivers".

2.1.4. Jun 16, 2021 Virtual F2F SAT

https://confluence.automotivelinux.org/display/VE/Meeting+Agenda?preview=/29425784/44826835/2021-06-16%20SAT%20virt%20EG.pdf#MeetingAgenda-Jun16,2021VirtualF2FSAT
Using Linux subsystem as common device I/F



2.2. Definition & Approach of Common Device Interface

Next is summarized a need for common device I/F and approaches to achieve it. Some of those approaches have been already widely used in past and are used currently:

- Custom kernel level HAL

- Custom user space level HAL
- Common kernel subsystem API and/or common user space libraries used by VirtIO as HAL

other is proposed as a know-how:

- Entire VIRTIO Framework as a HAL

2.2.1. Goals of common device I/F

Here are listed goals of common device I/F availability, most of this was already discussed:

- Supply application vendors with vendor lock-in free platform.
    - In case of native platform this means that applications should be independent from the HW vendor.
    - In case of virtualized platform this still means that application should be independent from underlying physical HW platform, but in addition, they should be independent from hypervisor too.
- Have standardized I/F.
- Support backward compatibility.

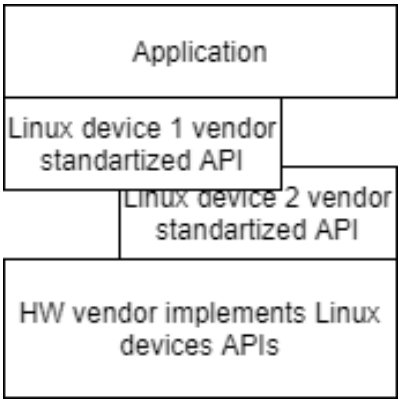
In addition, achieving above listed goals should bring extra advantages:

- Live in a healthier ecosystem
- Avoid fragmented efforts and waste of resources
- Improve debugging experience by having clearly defined interface. One can check if either supplier of user of that interface misbehaves to find a culprit.
- Improve debugging experience by having a possibility to substitute different device I/F implementations and compare result.

To achieve above stated goals, an important ingredient of a common device I/F is that it must be an open standard, i.e. it has to be free to implement by anyone, there should be possibility for anyone to participate in development of a future version of a common device I/F.

2.2.2. Alternatives to Define Common Device Interface

2.2.2.1. Custom standardized kernel level HAL



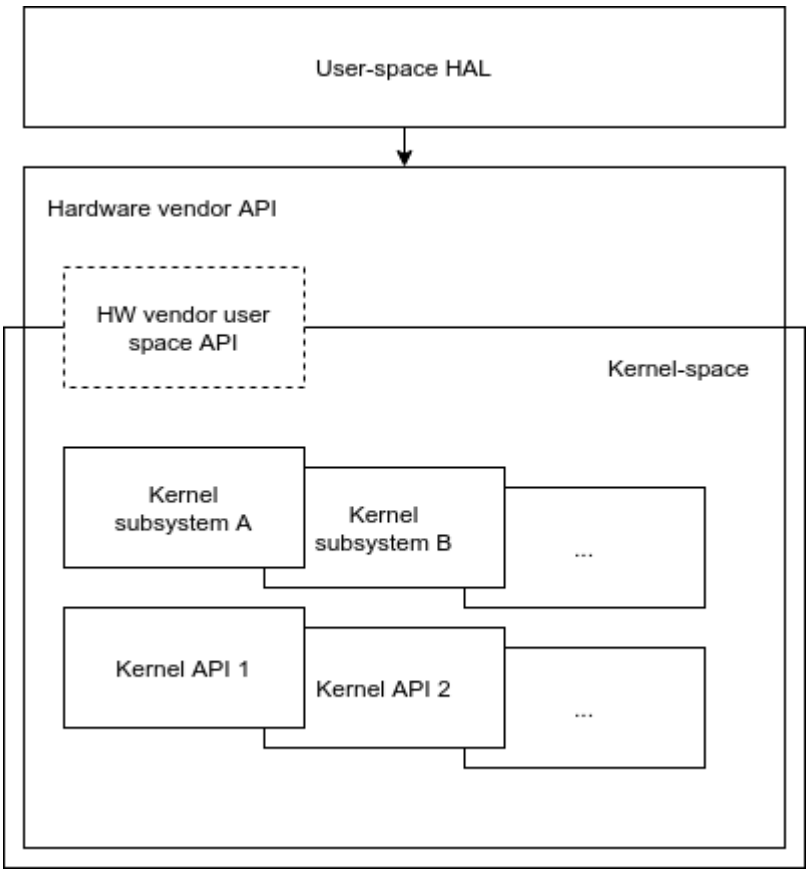
Here “Linux device N vendor standardized API” can mean for example Linux device node such as character or block device that provides standard syscalls such as:

- open
- read
- write
- mmap
- ioctl
- etc.

but semantics of those syscalls is HAL vendor specific. In many cases there is already kernel level device interface that is widely used and available in mainline kernel tree. But, instead of it, another HAL vendor-specific API might be proposed which comparing to common API might be standardized by a vendor.
This approach is adopted, for example, by some set-top-box software vendors where kernel level HAL specification can be approx. 2000 pages long.

2.2.2.2. Custom standardized user-space level HAL

This approach is adopted by Android:



It is needed if there is no single standardized kernel subsystem and/or kernel API which can act as a kernel space hardware abstraction layer for a device X.

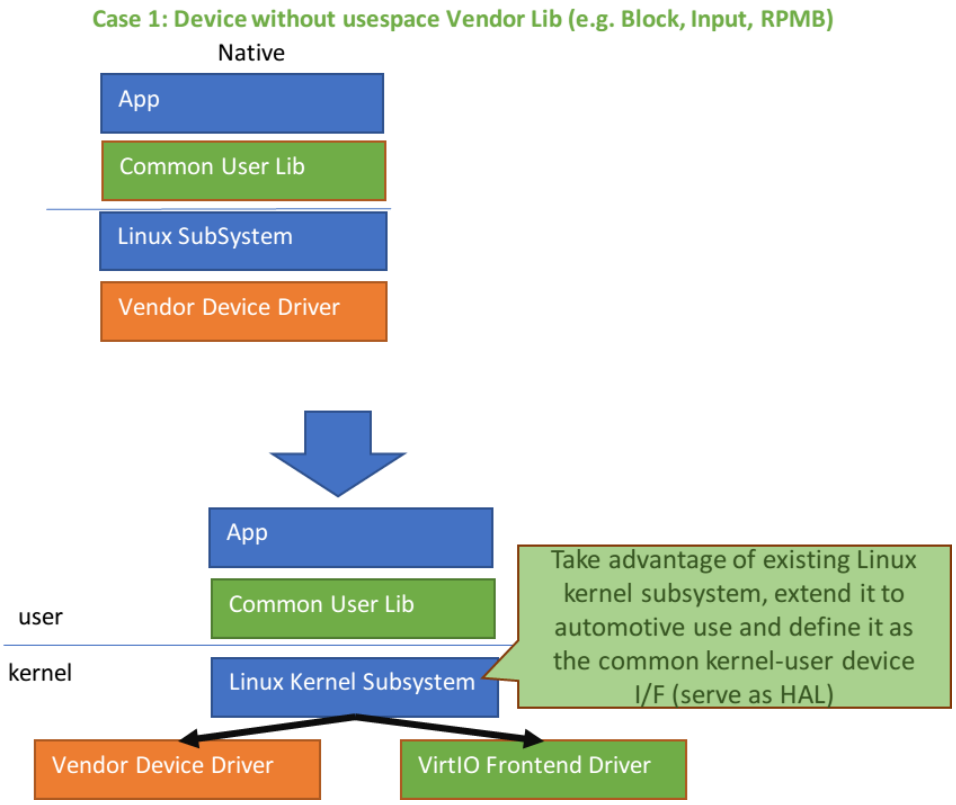
There are some disadvantages:

- applications have to be adapted for a particular user-space HAL
- since user-space HAL moves further away from biggest common denominator (e.g. “kernel”), there is less chance user-space HAL could achieve as much wide adoption as kernel-space HAL. For example, Android user-space HAL is not going to be used in ChromeOS, but both can use kernel-space HAL.

2.2.2.3. Common kernel subsystem API and/or common user space libraries used by VirtIO as a HAL

Original idea of common kernel subsystems or common user space libraries are abstracting hardware and use as a HAL. Unfortunately, the fact is that “custom kernel level HAL” and “custom user-space level HAL” approaches are still used by some vendors due to various reasons, which are described in the challenges section of this chapter.

However, common kernel subsystem API and/or common user space libraries can be used in either native or virtualized platform with minimal/near-zero changes when using VIRTIO for device virtualization. It is because that VIRTIO devices are already supported by common kernel subsystem APIs and common user space libraries.



Challenges when adopting common Linux subsystem as a HAL:

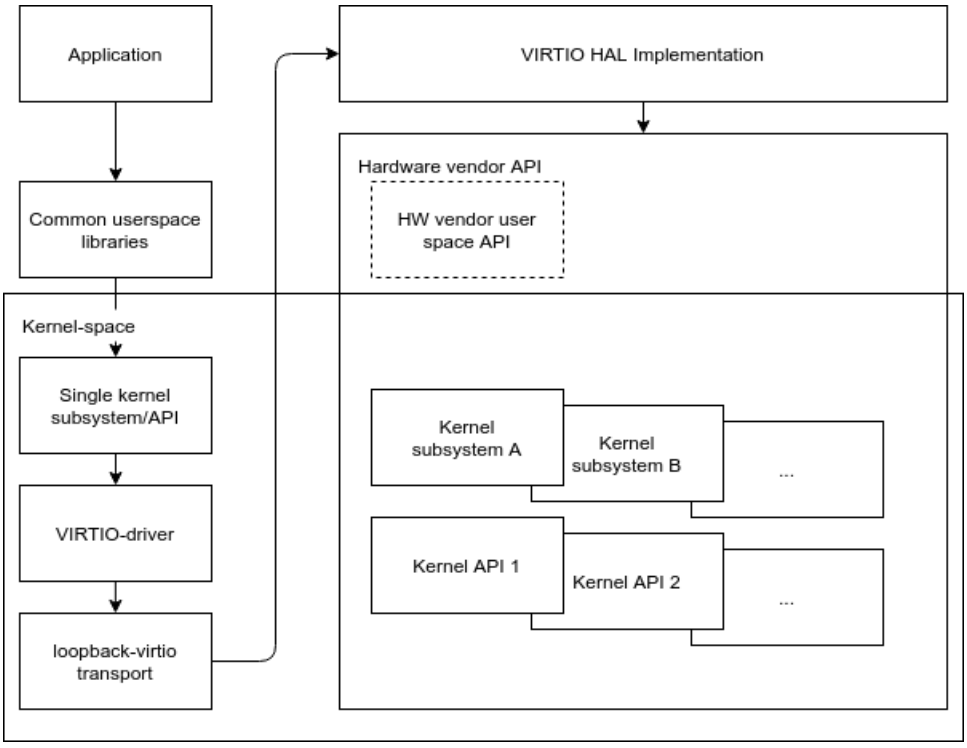
- There might be no agreed single Linux subsystem to abstract particular kind of device: for example, V4L2 vs OMX for codecs.
- Common Linux subsystem API might be not available for some HW
- Common Linux subsystem API might be not possible to implement for some HW by third-party without reverse engineering
- Backward compatibility:
    - https://github.com/torvalds/linux/tree/master/Documentation/ABI/ stable-level provides backward compatibility for at least 2 years and a guarantee that syscalls will never change.
    - On the other hand, devices with stable-level ABI are limited
- Standardization:
    - Similarly to above, there is https://github.com/torvalds/linux/tree/master/Documentation/ABI/stable but it is limited

Challenges when adopting common userspace libraries as a HAL:

- Might be no agreed common userspace libraries, for example gstreamer used by Linux but not Android
- Common userspace libraries might not be available for some HW
- Common userspace libraries might be not possible to implement for some HW by third-party without reverse engineering
- Backward compatibility:
    - Many libraries adopt https://semver.org and provide limited backward compatibility (within same major version).
- Standardization:
    - Usually, user library API is not separated from library itself.

2.3. Entire VIRTIO Framework as a HAL

Idea is to use VIRTIO standard as a hardware abstraction layer.



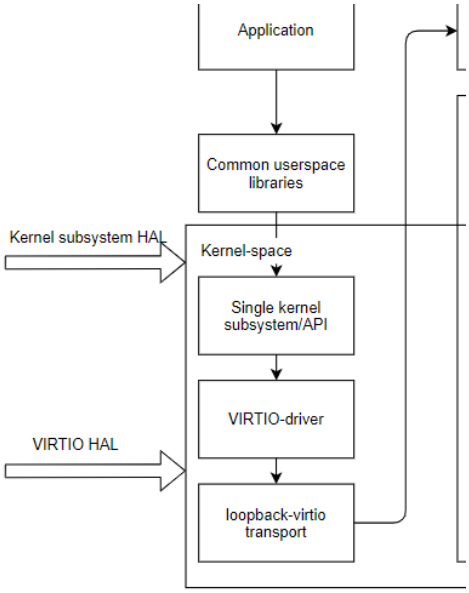
"Common userspace libraries" block is HW independent, but it should support VirtIO.

"VIRTIO HAL Implementation" block acts as "physical device driver adapted to the common device I/F in non-virt environment" from VIRT-003.

There are multiple advantages in reusing VIRTIO as HAL:

- Standard is already available; it is developed with a vendor neutral and operating system independent meaning
- VIRTIO supports both forward and backward compatibility
- Having standardized interface helps debugging:
    - one can check if top and/or bottom layer follow standard to find out culprit
    - one can substitute multiple interface implementations and compare behavior
- VIRTIO technical committee promised to deliver new spec version every 12 to 16 months

- VIRTIO is already widely used in the industry such as cloud (server world) and desktop (Chrome OS)
- VIRTIO is easy to extend: just send patch to virtio mailing list
- VIRTIO devices can be sandboxed to improve system security
- Flexibility in terms of different Linux subsystem support: VIRTIO as HAL resides on lower level comparing to Linux subsystem as a HAL



Currently VIRTIO drivers for VIRTIO devices implement of most common at the moment Linux subsystem for a particular device. But, in future, for a VIRTIO device different drivers might exist implementing different Linux subsystem APIs.

VIRTIO challenges:

- Additional performance overhead (should be negligible)
- Not all devices are standardized (although VIRTIO is actively developing)
- Additional effort to implement VIRTIO device (but already existing implementations can be reused).

2.4. Conclusion of Virt-EG

As conclusion of Virt-EG, in order to construct a more common device interface in dependent from SoCs, Hypervisors, Execution Environment (virtualized environment or native environment), Virt-EG would like to evaluate adopting the following two approaches as attempts:

1. Common kernel subsystem API and/or common user space libraries used by VirtIO as a HAL
2. Entire VirtIO Framework as a HAL

3. Requirement Specification

3.1. External interface requirements

- VIRTIO spec v1.1
- Kernel version: 5.4+
- AGL version: LL+

3.2. Performance requirements

- Follow zero-copy principle (tentative).
- Measure latency and throughput overhead of VIRTIO common device I/F implementation comparing to the case of common kernel subsystem API as a HAL without VIRTIO frontend/backend.
- For latency measurements use virtio-input
    - Generate input event using software input device (uinput)
    - Receive generated event via virtio-input. Measure time past.
- For throughput measurements use virtio-blk.

3.3. Functional requirements

In scope of this RFQ only following approaches to have common device I/F are relevant:

Approach 1: Common kernel subsystem API and/or common user space libraries used by VirtIO as HAL

Approach 2: Entire VirtIO Framework as a HAL

3.3.1. Functional partitioning

Function	Description	Related Approach
Touch sensitivity support	As a predefined feature for IVI-PR, touch panel should be able to support sensitivity get/set. Regarding approach 1 “common kernel subsystem API and/or common user space libraries used by VirtIO as HAL”, the intention is to demonstrate use of Linux subsystem as common device I/F by extending input subsystem with sensitivity support to accommodate HAL requirements. Regarding approach 2 “Entire VirtIO Framework as a HAL”, extending virtio-input framework (frontend and backend) to support sensitivity features is nessary.	Approach 1 & 2
virtio-loopback transport driver	Comparing to other transport drivers (such as virtio-mmio or virtio-pci), virtio-loopback transport driver is designed for use non-hypervisor environment. Only software implementation of VIRTIO drivers is needed. For interface to user-space device node **/dev/virtio-loopback** should be used	Approach 2
virtio-loopback adapter to vhost-user protocol	This adapter allows reuse existing hypervisor and virtual machine monitors independent VIRTIO devices that follow vhost-user protocol.	Approach 2
virtio-devices integration in native AGL	As a prototype, only three most basic virtio-devices are expected to be integration: virtio-blk, virtio-rng, virtio-input(*). * need extension to support sensitivity	Approach 2

3.3.2. Functional description

Touch sensitivity support

Some touchscreen displays support configuring display sensitivity level. For example, Samsung phones have: Settings > Display > Sensitivity > ON/OFF. Setting it to ON increases touch screen sensitivity. This feature is needed when protective case is used for phone.

Regarding approach 1 “common kernel subsystem API and/or common user space libraries used by VirtIO as HAL”, if it is possible to find touchscreen with configurable sensitivity for hardware used, then sensitivity configuration support could be added to input subsystem. Touchscreen driver should be extended to implement new sensitivity configuration API. Since touchscreen devices are not usually very complicated, such modification might be doable even by a non-hardware vendor.
For adaptation to a virtualized setup, VIRTIO input spec should be extended with sensitivity support. And, virtio-input driver should be extended to support new feature of the spec.

Regarding approach 2 “Entire VirtIO Framework as HAL”, besides virtio-input front should be extended with sensitivity configuration support, the virtio-input device backend should also be extended to support this configuration option. Comparing with approach 1, to implement sensitivity in this approach, no modification of physical HW driver is needed.

virtio-loopback transport driver

- virtio-loopback driver should be implemented as external kernel module.
- Source code could be pushed to meta-agl-devel/meta-egvirt layer as a locally stored source code together with BitBake recipe.
- On module init, it should register virtio_loopback_driver platform driver. This platform driver could match "virtio,loopback" compatible string. Corresponding DT node could be generated at runtime using DT overlay. TODO: Is DT entry mandatory for platform drivers? How about x86?
- virtio-loopback driver should create "/dev/virtio-loopback" device node.

- virtio-loopback driver should support IOCTL to realize virtio-device. Call to this IOCTL by virtio-loopback-vhost-user-adapter should result in "register_virtio_device" called.
- It is not clear at the moment if "/dev/virtio-loopback" should be single device node for all virtio-loopback devices or there should be /dev/virtio-loopback0, /dev/virtio-loopback1, …

virtio-loopback adapter to vhost-user protocol

- On the one hand, it should use "/dev/virtio-loopback" device node using read/write/IOCTL.
- On the other hand, it should communicate over Unix socket using vhost-user protocol.

It is not clear at the moment if adapter instance should be single for all virtio devices or there should be multiple instances of user space process running.

virtio-devices integration in native AGL

1. agl-demo-platform-virtio-native image should be created which extends standard agl-demo-platform image with virtio devices abstraction limited for BLK, RNG, INPUT.
2. This should involve installing additional drivers and user space daemons, configuring kernel, configuring systemd services.
3. Device specified function requirements:

   virtio-blk integration in native AGL

   - To realize virtio-blk on native for rootfs, it is needed to use initrd.
   - Boot should start from initrd which should start virtio-blk backend which consumes physical block device. As result, "/dev/vda" should be created and boot should continue with "/dev/vda" as rootfs.

   virtio-rng integration in native AGL

   - As an example of flexibility, recent virtio-rng implementation in Rust is proposed. For this, meta-rust should be integrated in AGL as a prerequisite to build virtio-rng backend.

virtio-input integration in native AGL

- virtio-input has to be configured to provide input events to applications from an available physical input device
- As described above, sensitivity configuration should be supported

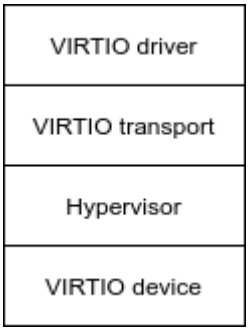
4. Hardware Structure

- Renesas R-Car H3 SoC based AGL Reference Hardware board: https://wiki.automotivelinux.org/_media/eg-rhsa/rh_manual_ver.1.1.pdf
- Touch-panel supporting configuration of sensitivity

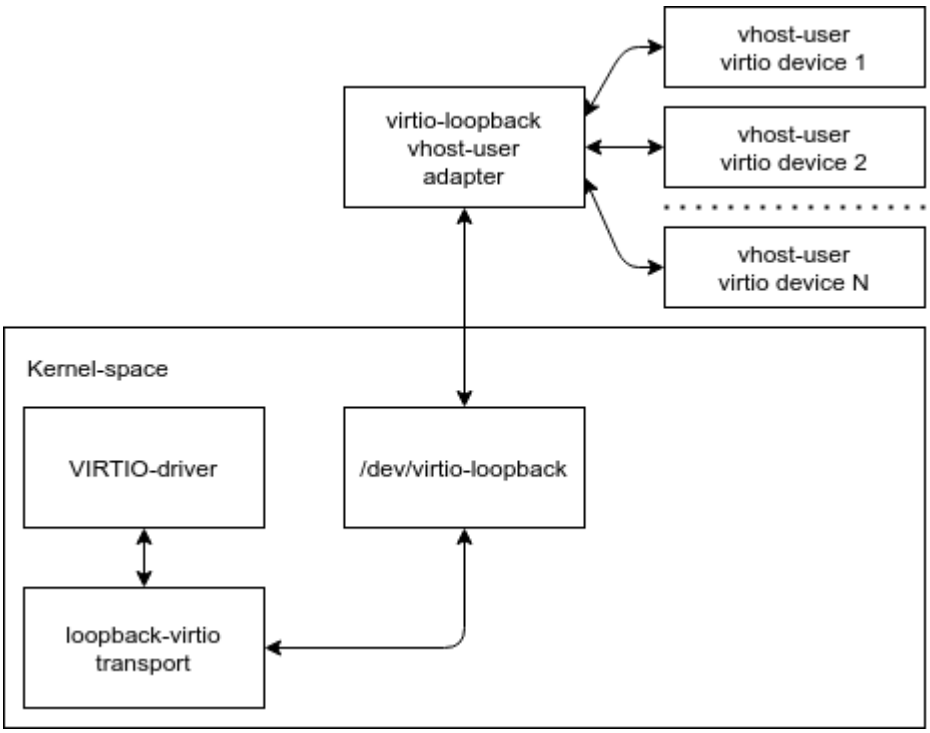
5. Software Structure

Linux input subsystem user-space API is defined in
`include/uapi/linux/input.h`
Control of device parameters are usually done using ioctl syscall, but in case of input device it might be controlled using event interface
https://www.kernel.org/doc/html/v5.14/input/input.html#event-interface
So, to control touch sensitivity, new event type can be allocated
`EV_TOUCH_SENSITIVITY`
see https://www.kernel.org/doc/html/v5.14/input/event-codes.html#event-types for list of existing event types. Handling of such event in specific touchscreen driver can be done by setting up `event` callback in `input_dev` structure.

Under control of hypervisor software implementation of VIRTIO devices is transparent to VIRTIO transport such as virtio-mmio or virtio-pci. VirtIO transport just pokes corresponding registers and handles interrupts. Register access is trapped by hypervisor, interrupts (virtual) are generated by hypervisor too.



In non-hypervisor environment a special VIRTIO transport has to be implemented which will explicitly forward VIRTIO requests to software implementation. Usually, software implementation is easier in user-space. In addition, there are several implementations of VIRTIO device in user space available and more are in development. In particular, there are VIRTIO devices implementation that use vhost-user protocol and therefore are independent from particular hypervisor or virtual machine monitor. Adapting virtio drivers and vhost-user devices in non-hypervisor environment is illustrated on the following diagram:



Implementation of specific VIRTIO devices should not be part of this PoC, instead available open-source vhost-user VIRTIO devices shall be used. Below list is used as an illustration, it does not mean that all devices have to be part of this PoC.

virtio-blk	https://github.com/qemu/qemu/blob/master/hw/block/vhost-user-blk.c
virtio-fs	https://github.com/qemu/qemu/blob/master/hw/virtio/vhost-user-fs.c
virtio-gpu	https://github.com/qemu/qemu/tree/master/contrib/vhost-user-gpu
virtio-i2c	https://github.com/rust-vmm/vhost-device/tree/main/src/i2c
virtio-rng	https://github.com/mathieupoirier/vhost-device/tree/vhost-device-rng-v2/src/rng
virtio-scmi	https://github.com/Gaelan/vhost-device/tree/scsi-v1/src/scsi
virtio-vsock	https://github.com/harshanavkis/vhost-device/tree/vsock/src/vsock
virtio-input	https://github.com/qemu/qemu/blob/master/contrib/vhost-user-input/main.c

6. Activities & Excepted outcome

[Approach-1-Related] Extend Linux input subsystem with touchscreen sensitivity configuration API.

Patch to LKML that extends Linux input subsystem with touchscreen sensitivity configuration API. It is possible to do only if there can be found a hardware that supports this and implements such API, otherwise patch will not be acceptable.

Patch has to be backported to

`meta-agl-devel/meta-egvirt/dynamic-layers/meta-agl-bsp/meta-PLATFORM`

layer, where PLATFORM is a particular platform for which touchscreen with configurable sensitivity can be found.

[Approach-1-Related] Extend particular physical touchscreen hardware driver with a support of Linux input subsystem touchscreen sensitivity configuration API.

If it is possible to find touchscreen hardware with sensitivity configuration that is possible to connect to a board supported by AGL. Its driver has to be extended to implement introduced above sensitivity configuration API.

If driver is available in mainline kernel, patch to LKML has to be submitted.

Patch has to be backported to

`meta-agl-devel/meta-egvirt/dynamic-layers/meta-agl-bsp/meta-PLATFORM`

[Approach-1&2-Related] Extend virtio-input spec and driver with support of touchscreen sensitivity configuration API.

It has to be investigated whether it is needed. Virtio-input might already support configuration via event interface. So, as soon as event type to configure touchscreen sensitivity is introduced, it will be supported by virtio-input.

[Approach-2-Related] Design virtio-loopback device interface.

virtio-loopback device interface consists of syscalls (read, write, mmap, ioctl, etc.) for virtio-loopback device to expose virtio transport to userspace.

It has to be investigated how much vhost device interface

`include/uapi/linux/vhost.h`

can be reused.

[Approach-2-Related] Develop virtio-loopback virtio transport driver.
virtio-loopback driver acting as virtio transport on the one hand, and as an implementation of virtio-loopback device on the other hand, has to be implemented and submited to LKML.
Patch has to be backported to
`meta-agl-devel/meta-egvirt/dynamic-layers/meta-agl-bsp/meta-PLATFORM`

[Approach-2-Related] Develop virtio-loopback device to vhost-user adapter application.
Application has to be developed to connect virtio-loopback device and vhost-user backend available via Unix domain socket. It should be investigated how much vhost-user protocol can be transferred to/from virtio-loopback as is. This application has to implement vhost-user protocol “master” part.
Application has to be merged to
`meta-agl-devel/meta-egvirt`

[Approach-2-Related] Virtio-{blk,rng,input} integration.
Virtio-{blk,rng,input} have to expose physical devices in case of using
`agl-demo-platform-virtio-native`
image. Necessary changes have to be merged to
`meta-agl-devel/meta-egvirt`

[Approach-2-Related] Touchscreen sensitivity configuration using virtio-input backend.
In case there is touchscreen physical hardware available with configurable sensitivity, and it is possible to connect to one of the boards supported by AGL, then virtio-input backend has to be extended to configure touchscreen sensitivity.

[Approach-2-Related] Performance Overhead Measurement for native case
Performance overhead measurement for overhead caused by additional virito framework applied to native use case should be measured according “Performance Requirements” chapter and provide result report to Virt-EG

- Latency overhead measured for virtio-input.
- Throughput overhead measured for virtio-blk.

7. Target Schedule

The PoC activities target to complete by the end of CY.2022 but requires an intermediate PoC to be shown in the AGL All Member Meeting (AMM) or Automotive Linux Summit (ALS), which is typically held in Summer or Autumn, in order to show as a demo for Virt-EG activities in the event. The target AGL release is to be NN or OO (TBD).