跨操作系统的音频驱动模块设计与实现

答辩展示

穆新宇

北京理工大学计算机学院

2024 年 5 月 31 日

背景与动机

课题背景

在云计算时代，操作系统作为 Guest OS 运行在虚拟化框架上提供服务的情景非常常见，要使 Guest OS 能正确使用虚拟化框架提供的虚拟设备，就需要为这些虚拟设备编写驱动程序，例如，Linux 源码中就有很多虚拟设备驱动程序。

背景与动机

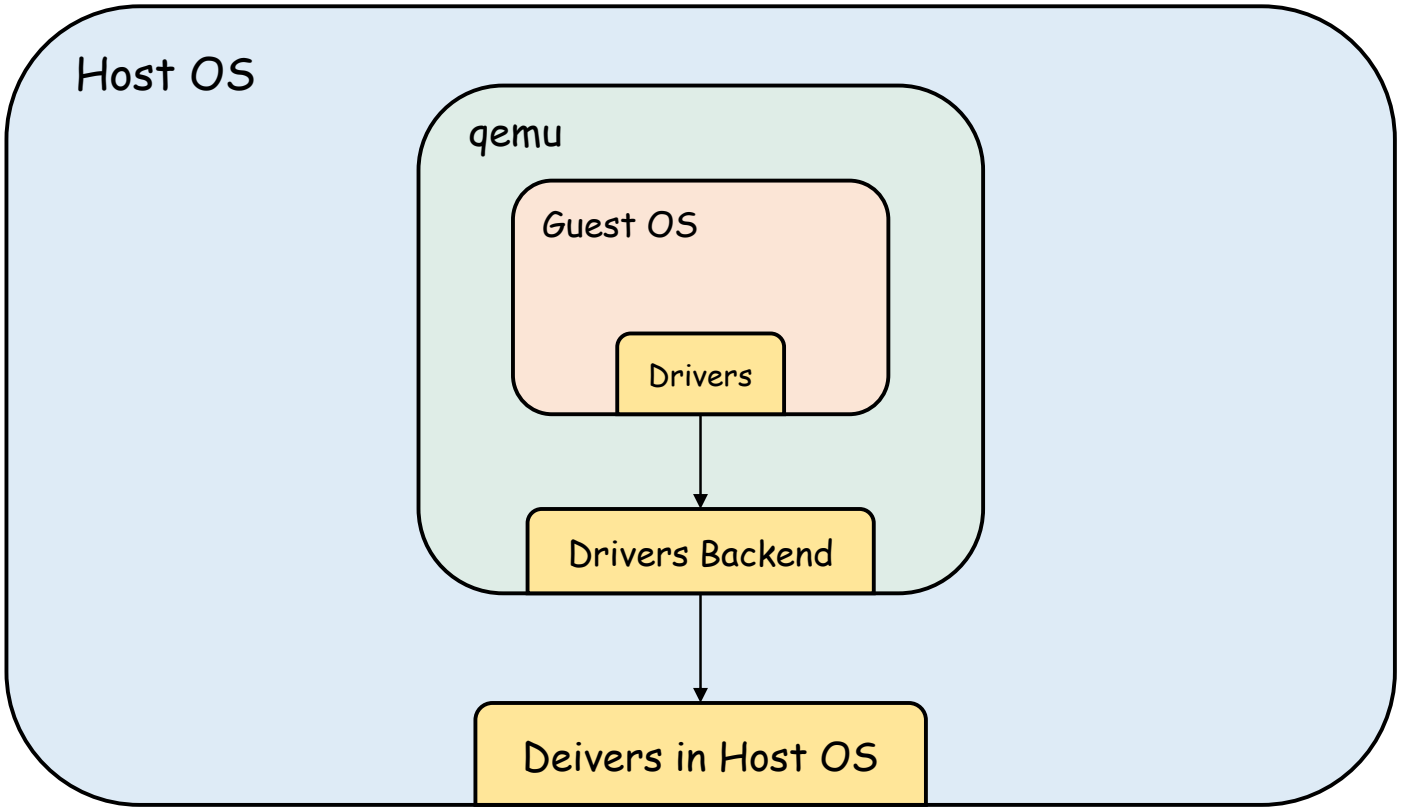


图 1: 在 qemu 上运行客户机时的驱动程序层级

背景与动机

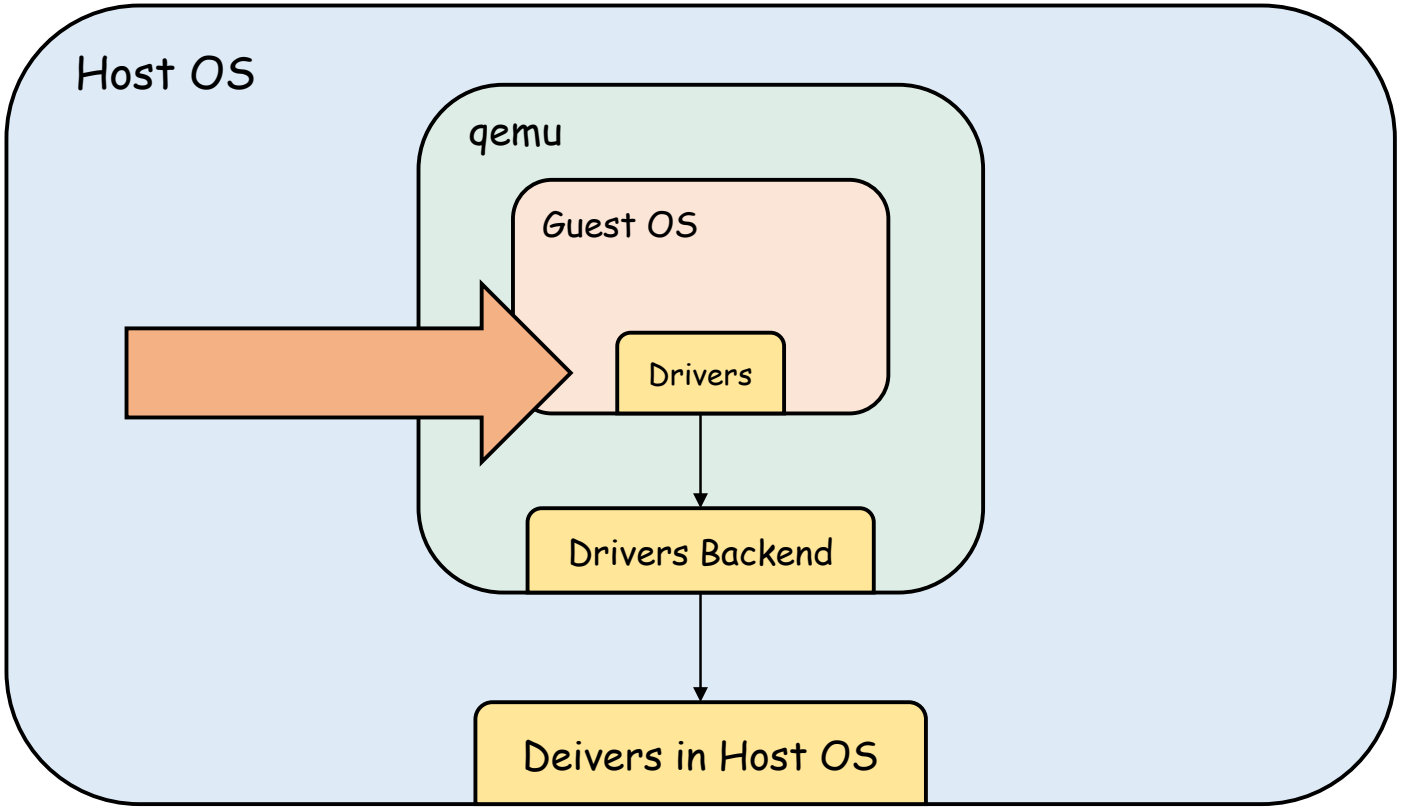


图 2: 客户机上的虚拟驱动程序

背景与动机

动机

每一个作为 Guest OS 的操作系统都要实现一遍 virtio 标准规定的驱动程序，对于一个新出现的操作系统，

- 很难直接利用已有的虚拟驱动程序实现
- 难以移植 Linux 源码中的实现

为此，一个跨操作系统的 virtio 驱动程序框架是必要的，这将极大减轻开发者的“重复造轮子”工作。

virtio-drivers

virtio-drivers 是使用 Rust 编写的跨操作系统的虚拟驱动程序框架，它参考了 Linux 源码中虚拟队列的实现，目前支持 block、net、input、gpu 等虚拟设备。

背景与动机

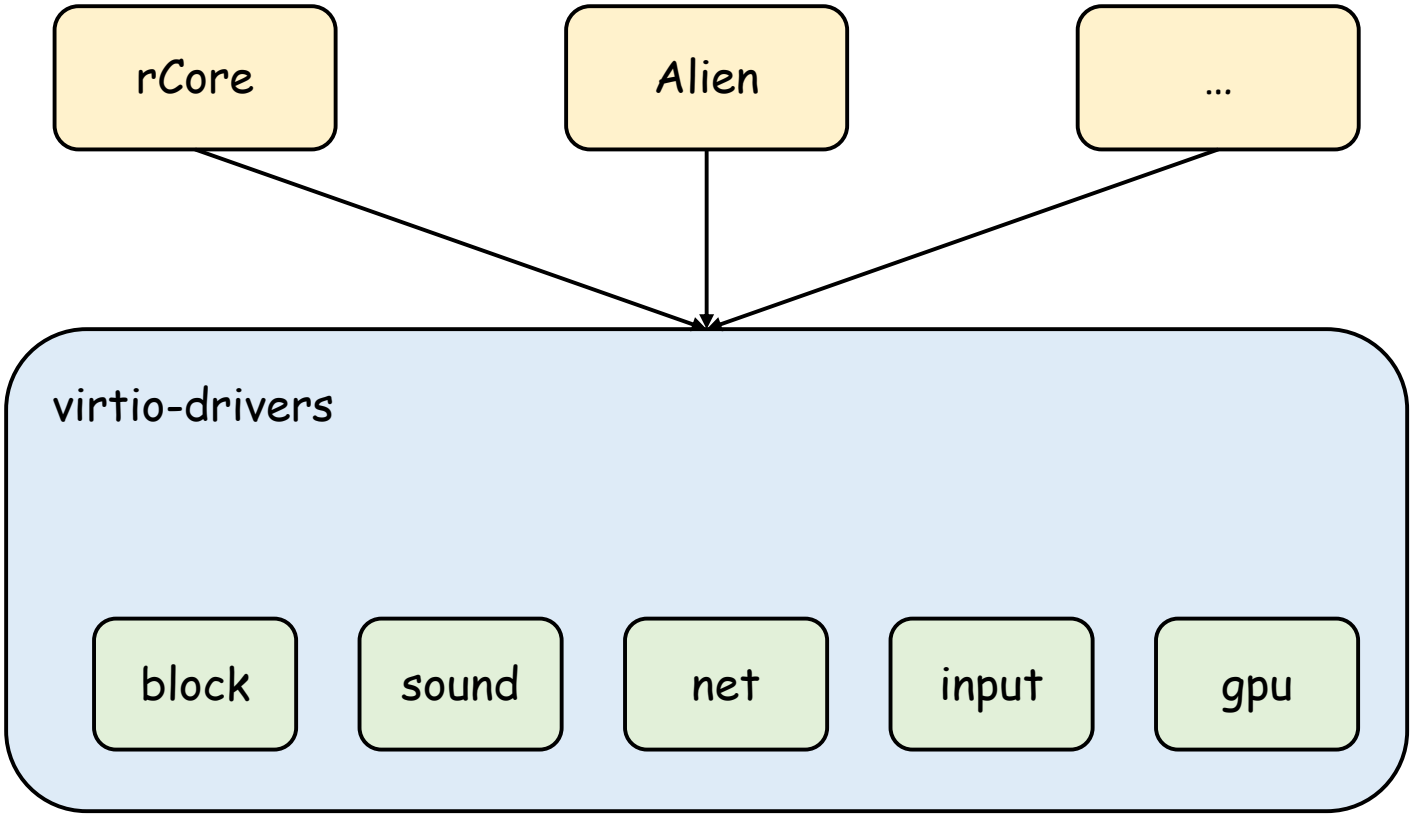


图 3: 使用 virtio-drivers 虚拟驱动程序框架

创新点

本项目在原有 virtio-drivers 框架基础上，

- 实现了跨操作系统的音频驱动程序
- 在裸机和 Alien 操作系统上进行了正确性验证

virtio

Rusty Russell 在探索虚拟机中设备驱动程序性能和效率的改进问题时提出了 virtio，virtio 是编写前后端驱动程序时需要遵守的协议，它一共有三层架构：

- 上层是运行的 qemu 等模拟器上的操作系统中的虚拟驱动程序
- 下层是 qemu 等模拟器中的虚拟设备
- 中间一层是传输层，作为驱动程序和虚拟设备交互的接口

在实现音频驱动程序时，中间层已经被 virtio-drivers 框架实现了，下层虚拟设备由 qemu 实现，所有我们只需**利用 virtio-drivers 提供的传输层，按照 virtio 协议与 qemu 提供的虚拟设备交互即可**。

virtio

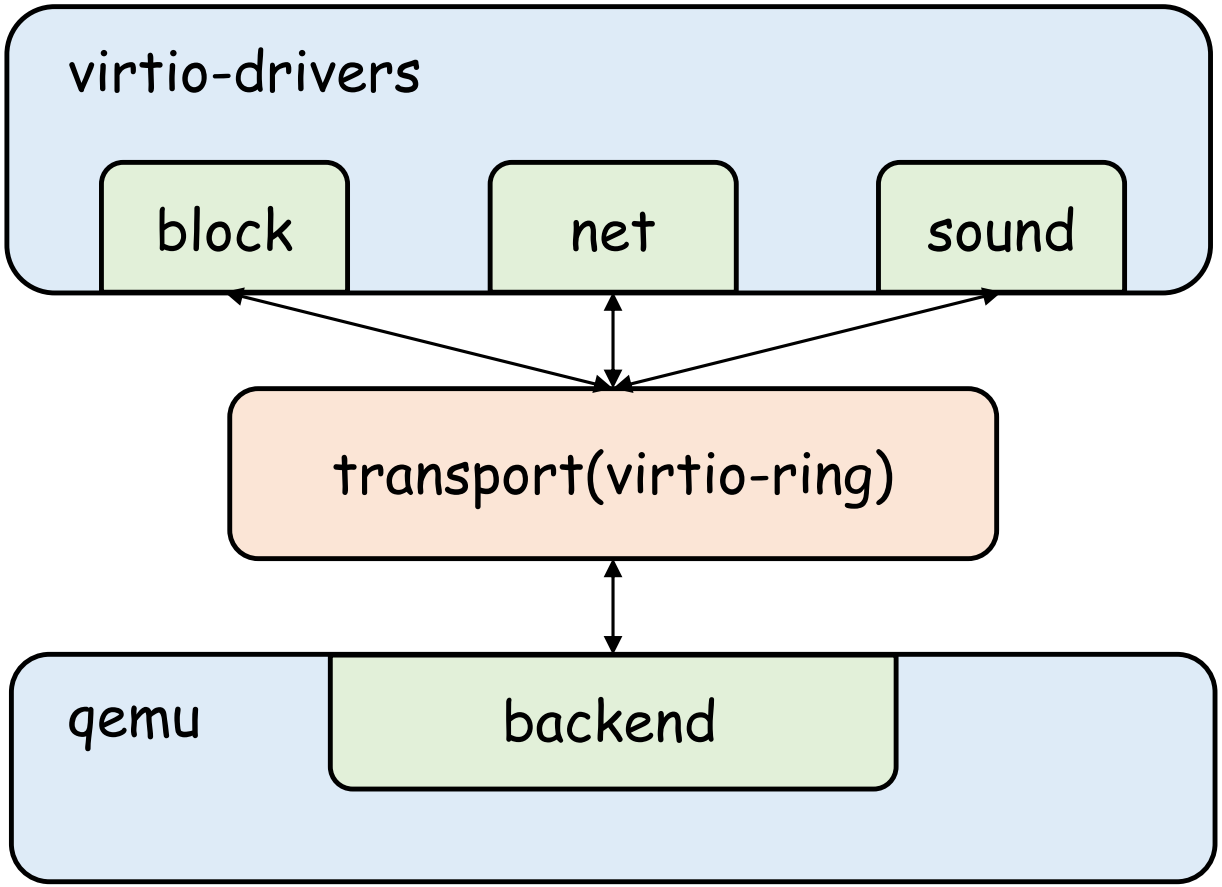
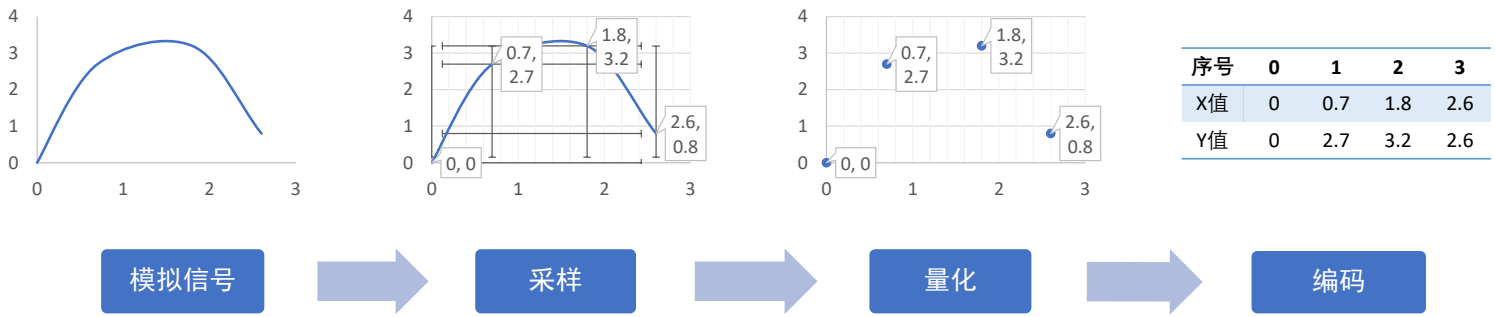


图 4: virtio 协议定义的三层架构

PCM

概念

脉冲编码调制（Pulse Code Modulation,PCM）是一种常见的将人耳听到的声音模拟信号转为数字信号的技术，主要分为**采样、量化、编码**这三个过程。

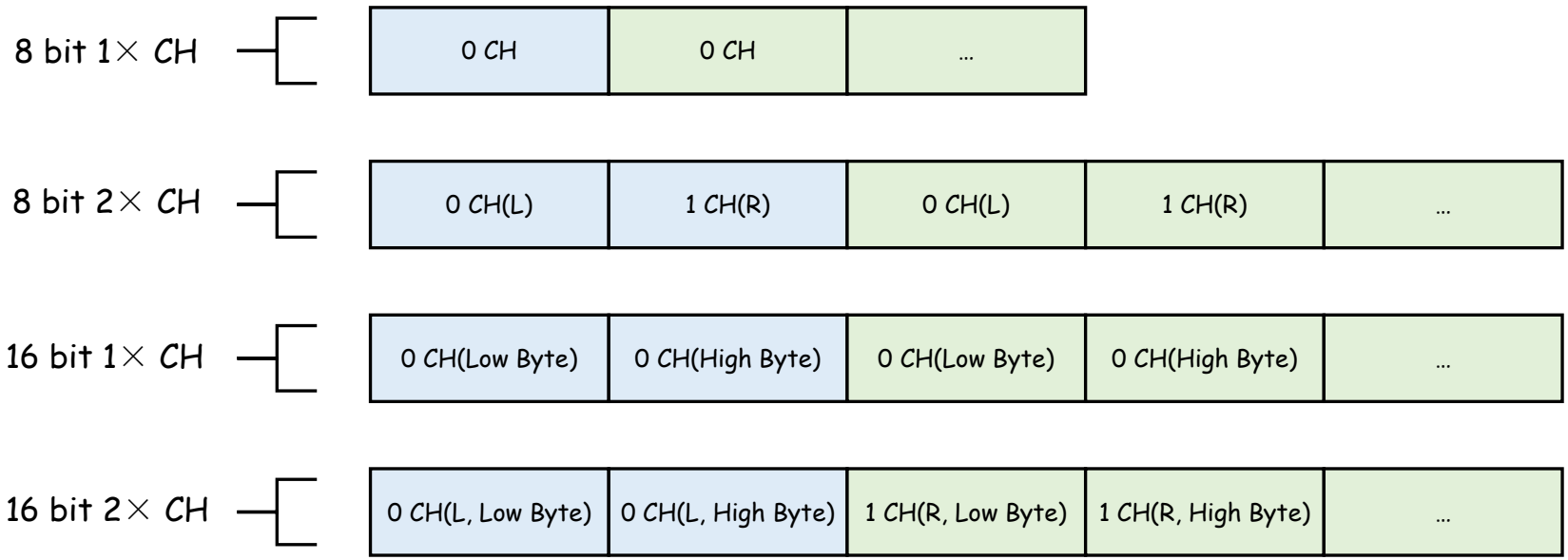


序号	0	1	2	3
X值	0	0.7	1.8	2.6
Y值	0	2.7	3.2	2.6

图 5: PCM 的三个过程

PCM

PCM 数据结构


8 bit 1× CH
0 CH
0 CH
...
8 bit 2× CH
0 CH(L)
1 CH(R)
0 CH(L)
1 CH(R)
...
16 bit 1× CH
0 CH(Low Byte)
0 CH(High Byte)
0 CH(Low Byte)
0 CH(High Byte)
...
16 bit 2× CH
0 CH(L, Low Byte)
0 CH(L, High Byte)
1 CH(R, Low Byte)
1 CH(R, High Byte)
...

整体架构

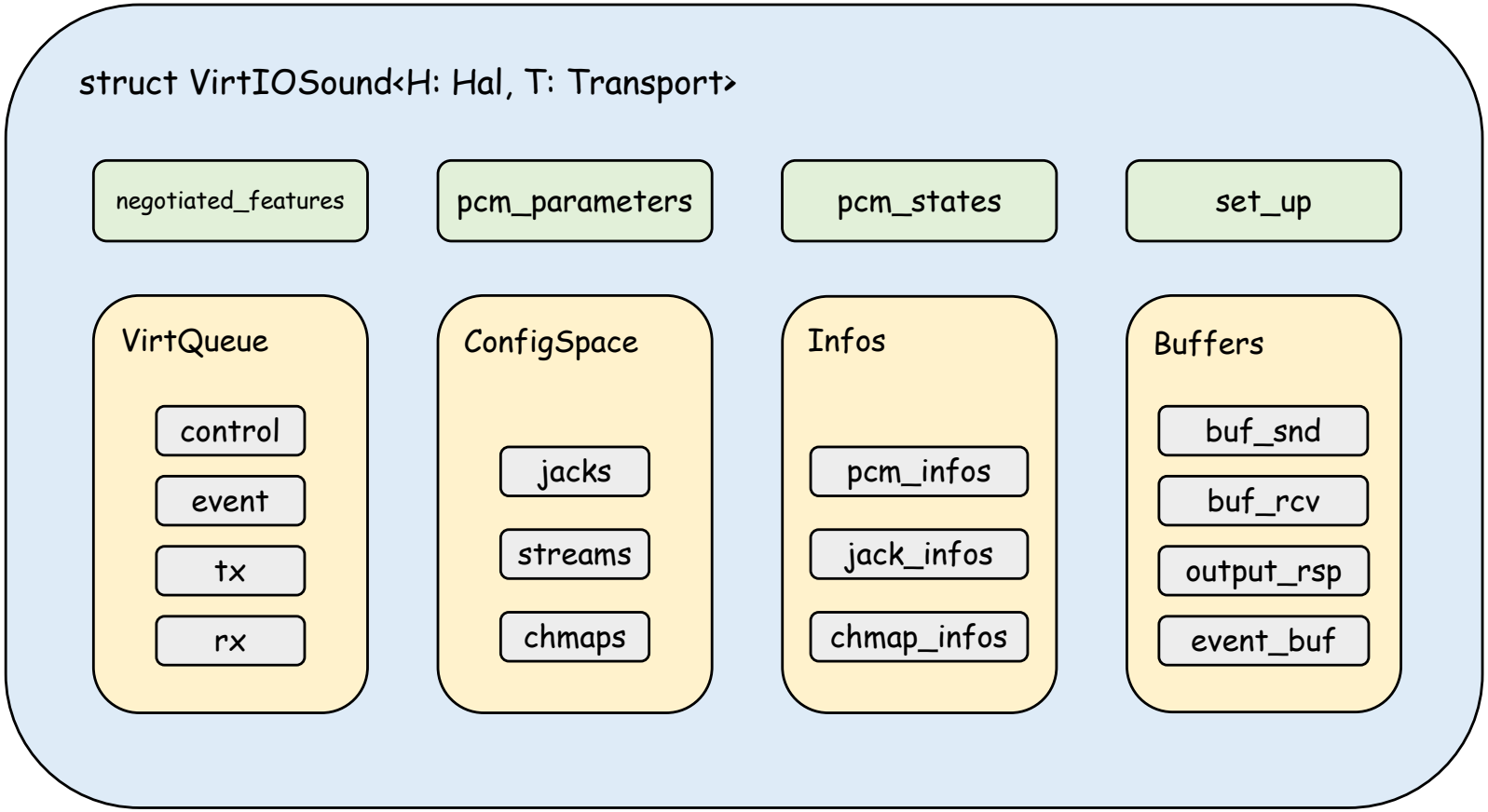


图 6: 驱动程序整体架构

整体架构

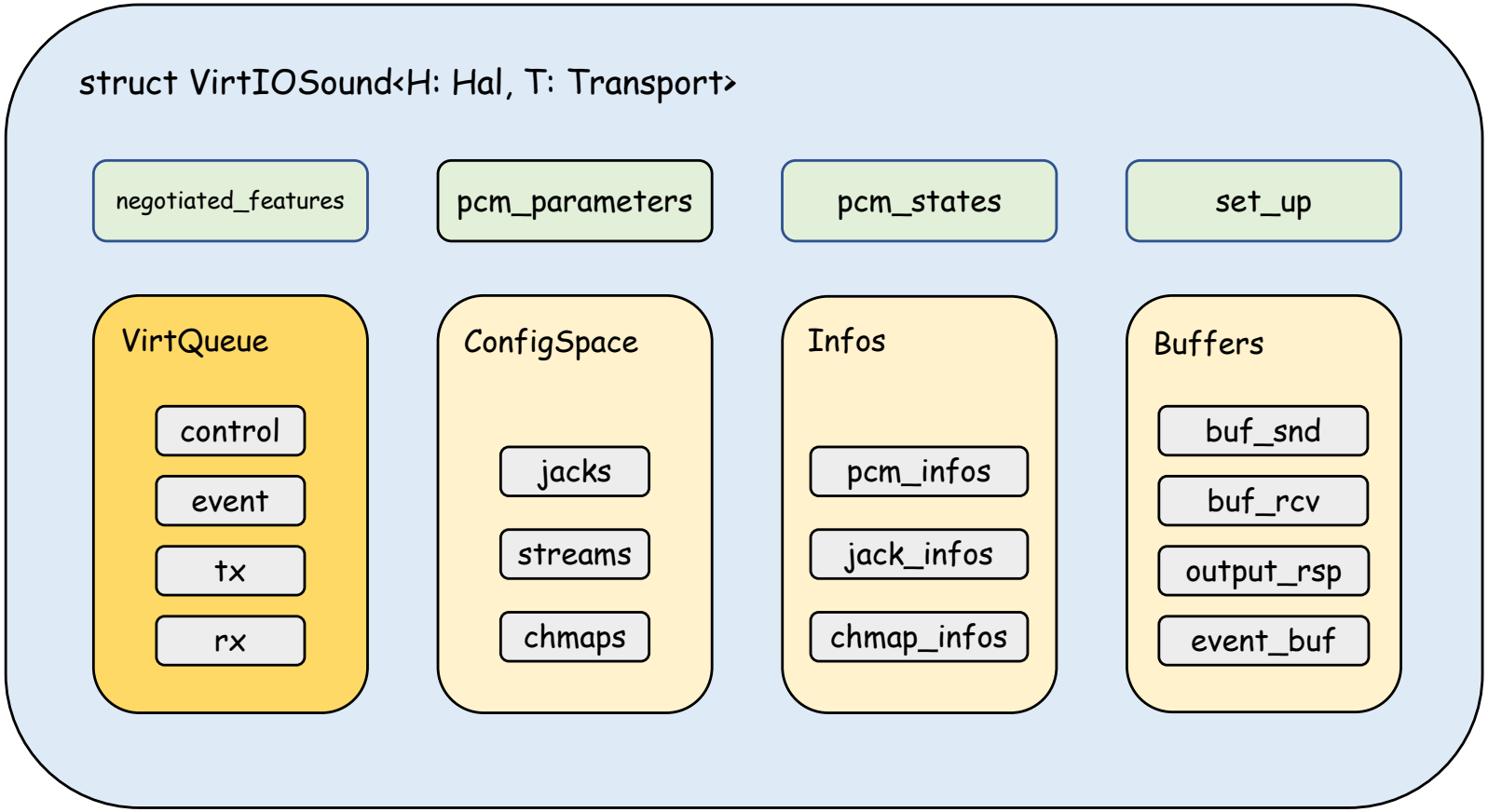
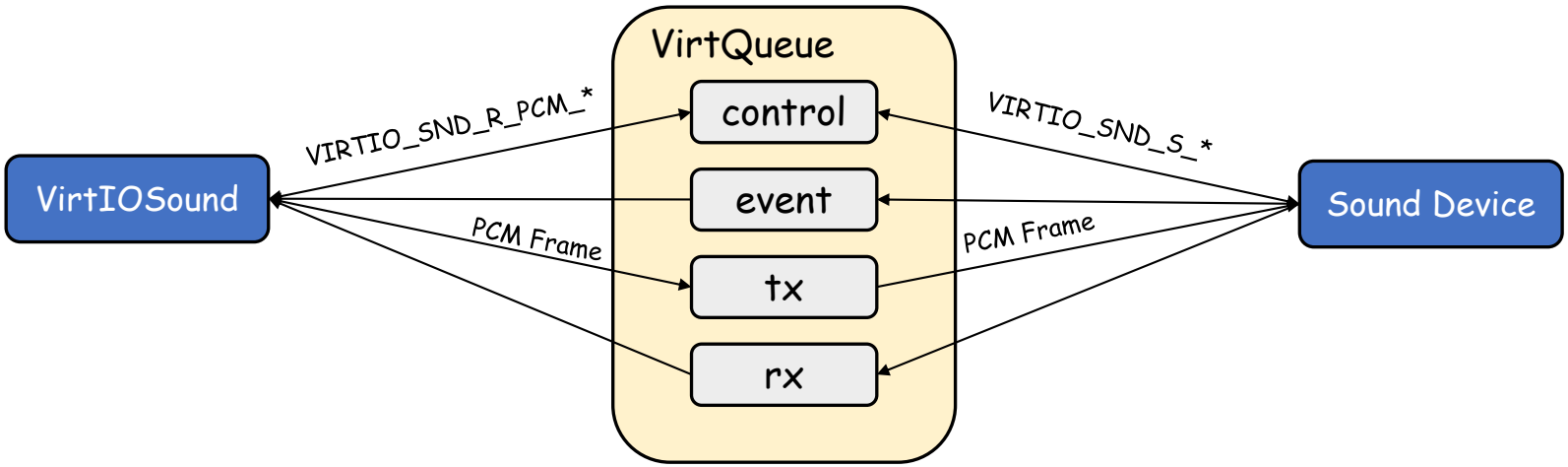

struct VirtIOSound<H: Hal, T: Transport>
negotiated_features
pcm_parameters
pcm_states
set_up
VirtQueue
control
event
tx
rx
ConfigSpace
jacks
streams
chmaps
Infos
pcm_infos
jack_infos
chmap_infos
Buffers
buf_snd
buf_rcv
output_rsp
event_buf

图 7: 虚拟队列

整体架构

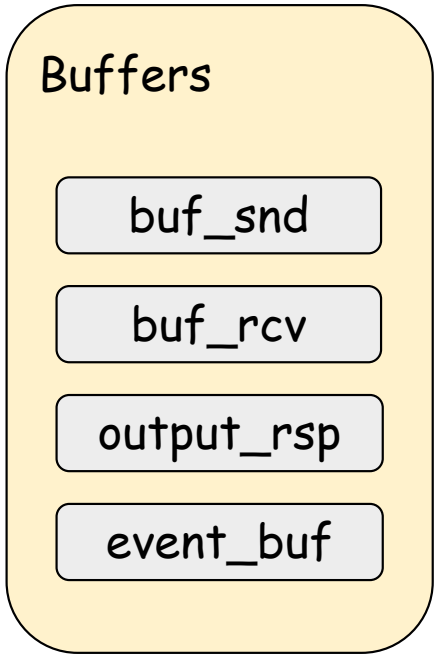
虚拟队列



- control_queue 用于从驱动程序向设备发送控制消息
- event_queue 用于驱动程序接受来自设备的通知
- tx_queue 用于向输出流发送 PCM 帧
- rx_queue 用于从输入流接收 PCM 帧

整体架构

Buffers



- buf_snd 存储驱动程序通过 control_queue 向设备发送的请求
- buf_rcv 存储设备通过 control_queue 向驱动程序发送的响应
- output_rsp 存储驱动程序每传输一个 PCM 帧，设备的响应
- event_buf 存储来自设备的通知

PCM 帧的传输

PCM 生命周期

PCM 生命周期有 7 个阶段，这 7 个阶段如下：

1. **SET PARAMETERS**：设置某 PCM 流的参数
2. **PREPARE**：PCM 流为传输做准备（资源分配等）
3. 仅输出：驱动程序在预缓冲时传输数据
4. **START**：设备开始播放/接收一个流
5. 驱动程序向流传输数据，或从流中接收数据
6. **STOP**：停止某 PCM 流
7. **RELEASE**：释放某 PCM 流

PCM 帧的传输

设备每使用一个缓冲区，就会将该缓冲区对应的 token 从 tx_queue 中弹出，然后从 tx_queue 中选择下一下缓冲区进行播放。因此，为了使音频播放不间断，tx_queue 中至少有两个缓冲区，如图8所示。

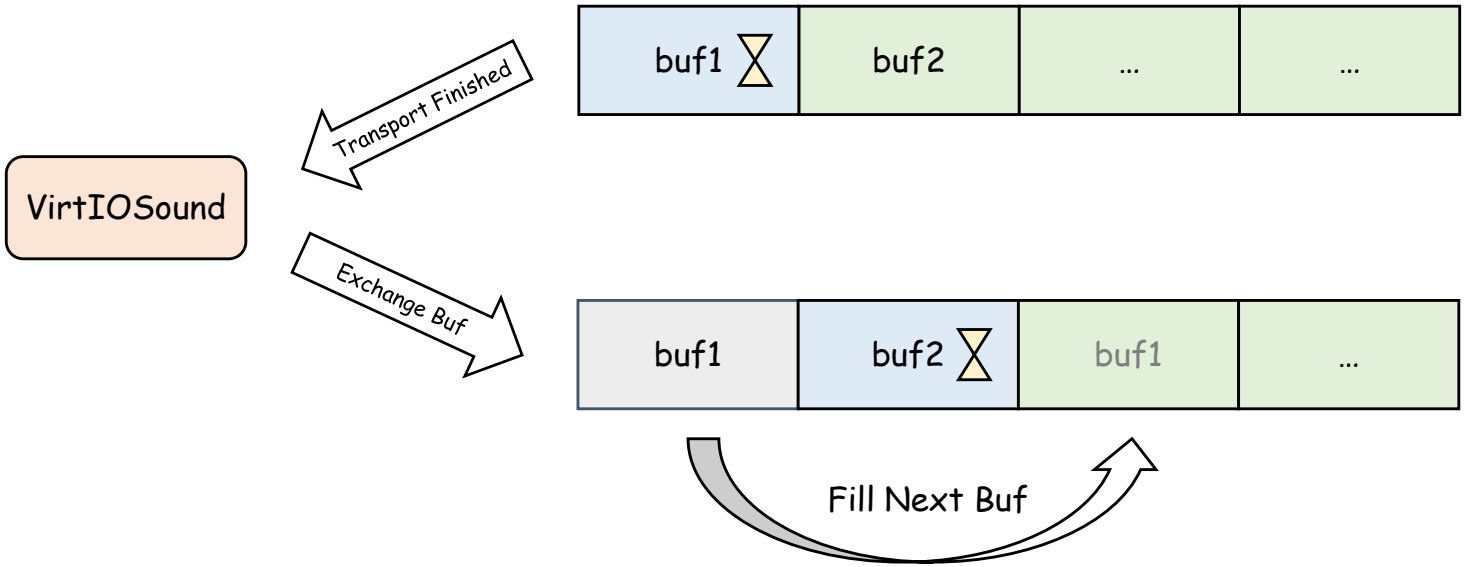


图 8: 双缓冲区交替

裸机

编写裸机测试程序，按照 PCM 生命周期的 7 个步骤配置 PCM 流，能够正确播放音乐：

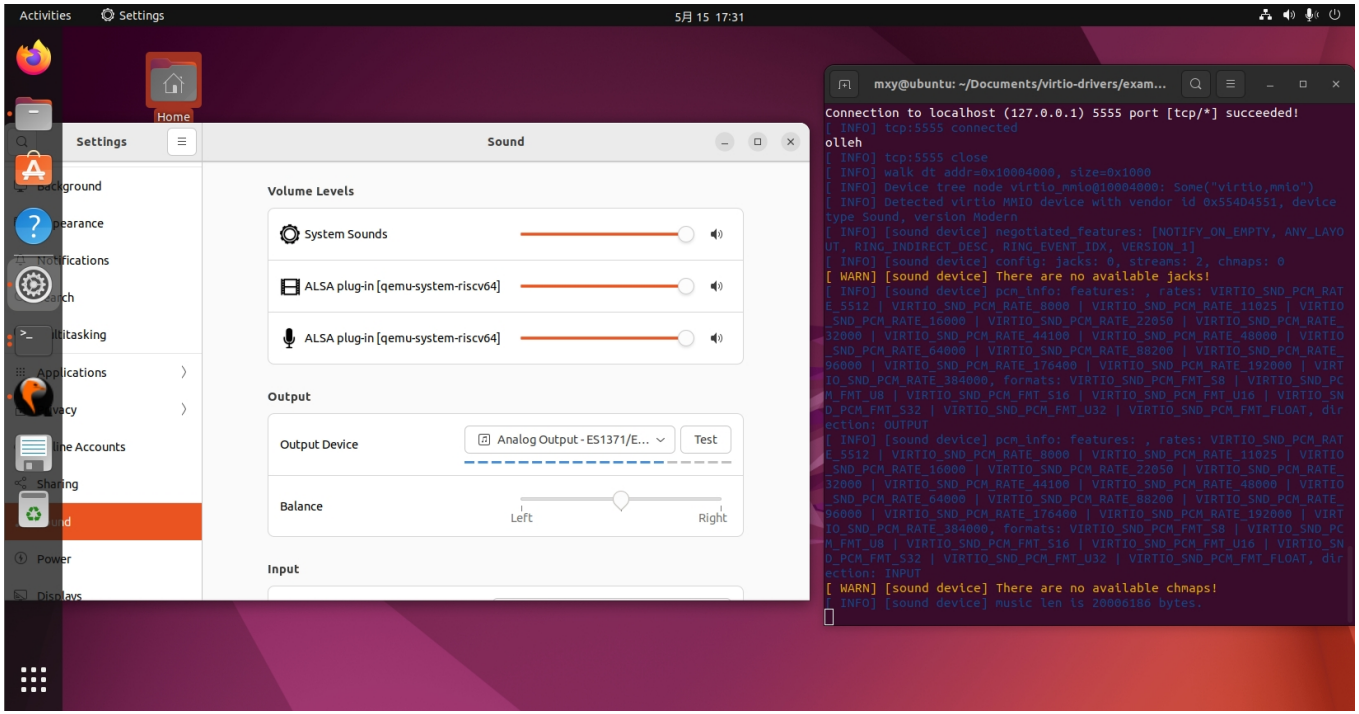


图 9: 裸机测试下正确播放了音乐

Alien 操作系统

在 Alien 文件系统中注册 Sound 设备，定义 SoundDevice trait，并为我们的驱动程序添加一层 Wrapper，就能够在内核态正确地播放音乐：

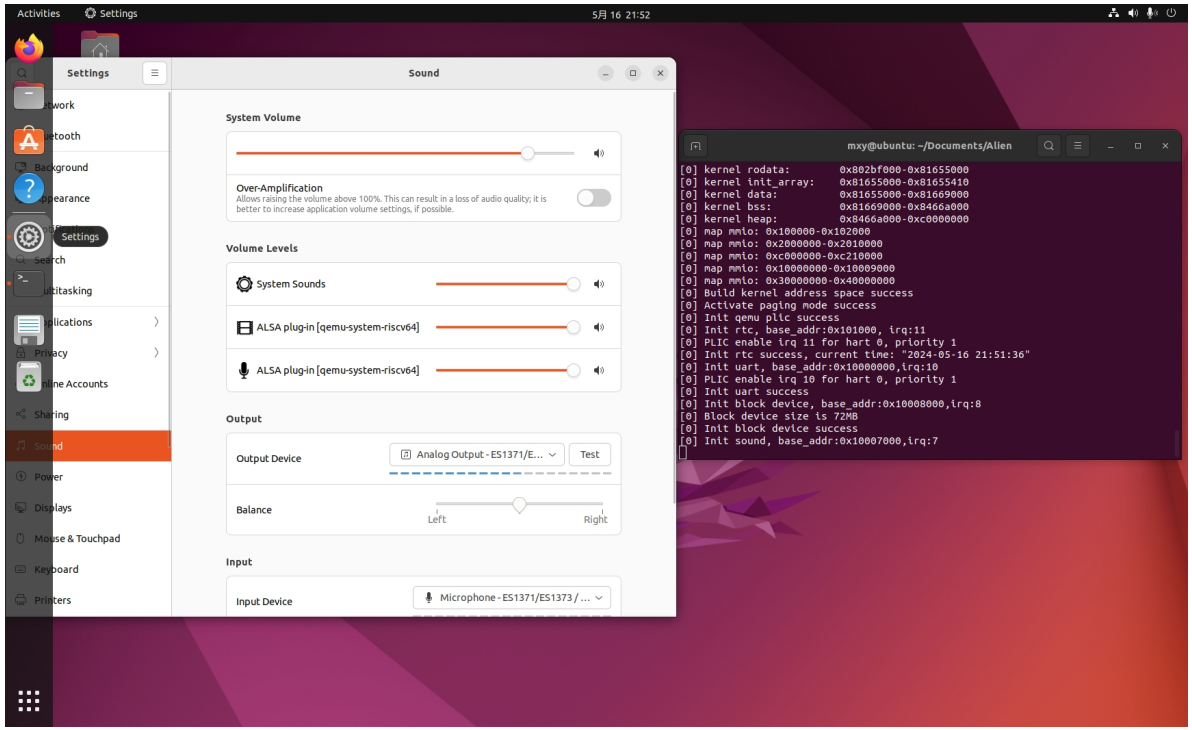


图 10: 在 Alien 中测试，正确播放了音乐

总结与展望

总结

总的来说，本项目利用 virtio-drivers 框架，按照 virtio v1.2 标准，实现了跨操作系统的音频驱动程序，并分别在裸机和 Alien 操作系统中验证了其正确性，为开源操作系统的发展贡献了自己的力量。

展望

本项目的最终实现虽然保证了正确性，也具有一定的鲁棒性，但是仍然存在不足之处，主要体现在：

1. 采用程序控制的方式，降低了 CPU 的效率
2. 实现了 virtio v1.2 标准，最新的标准是 virtio v1.3

这些缺点都是未来的工作中需要改进的地方。

谢谢！