winlator是如何让安卓手机跑windows 游戏的

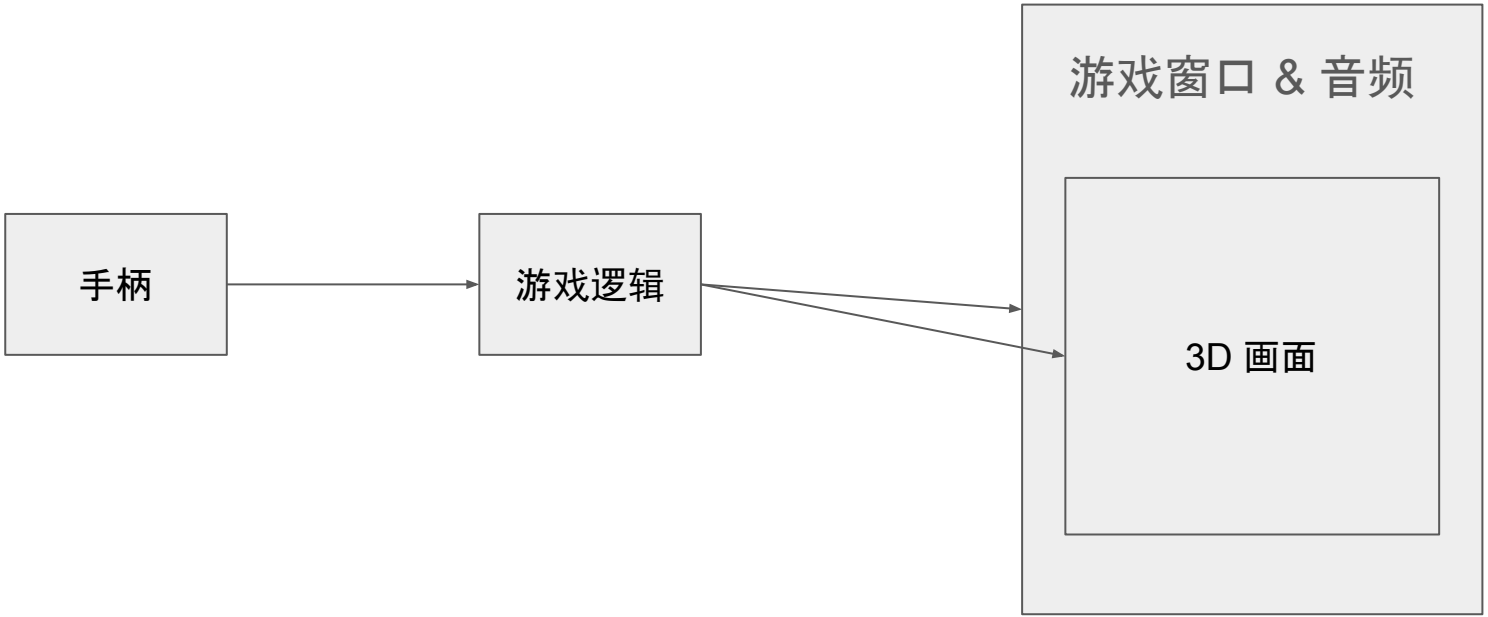
qq群 673249606

https://connect-screen.com/static/winlator-under-the-hood.pdf

普通用户管这些干啥？可能就是好奇吧……

- winlator-proot, winlator-glibc, winlator-bionic, mice-wine, 盖世游戏，这么多套方案？这些集成商底层技术是不是同一套？
- box64, dxvk, vkd3d, turnip 为啥用户需要调整这么多参数？这些参数不能出厂就设好么？为什么要给每个游戏调不同的参数？
- xinput, dinput, hid 都是啥？为什么我的手柄就是不识别？

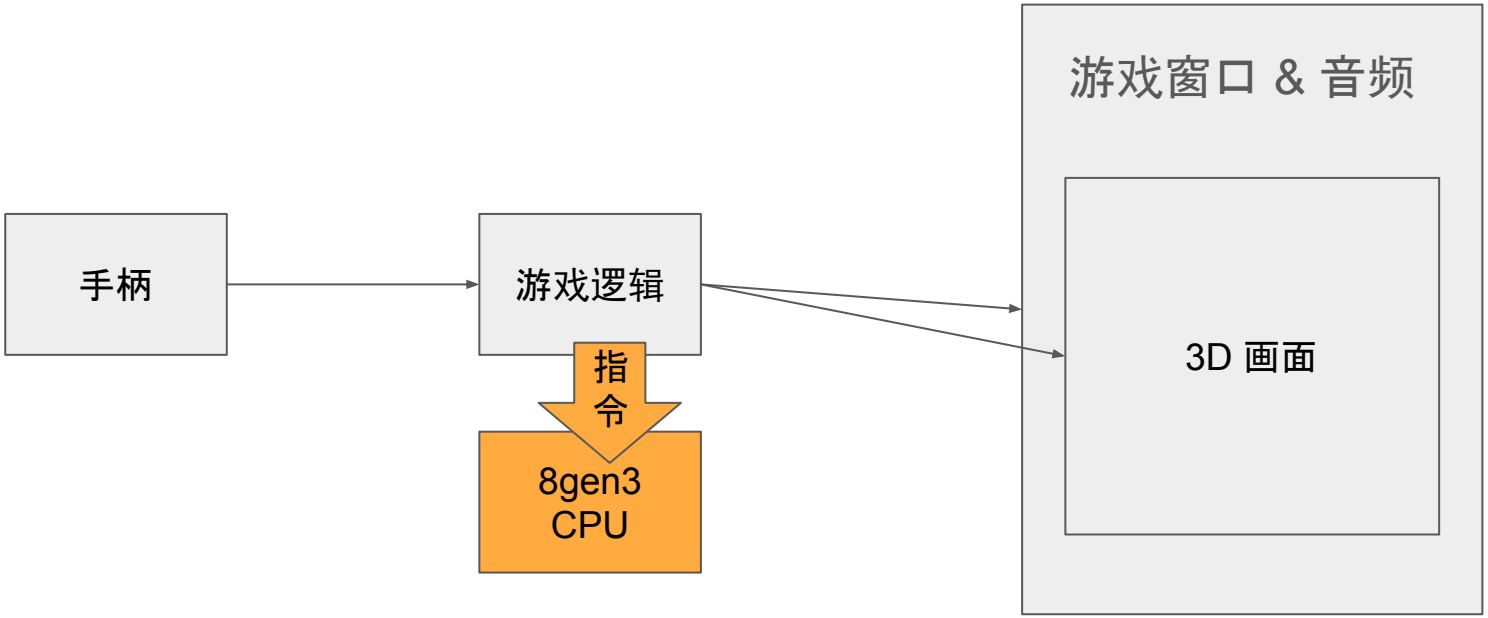
游戏的构成



高通 8gen3 在安卓上运行哈迪斯 2 要克服哪些困难？

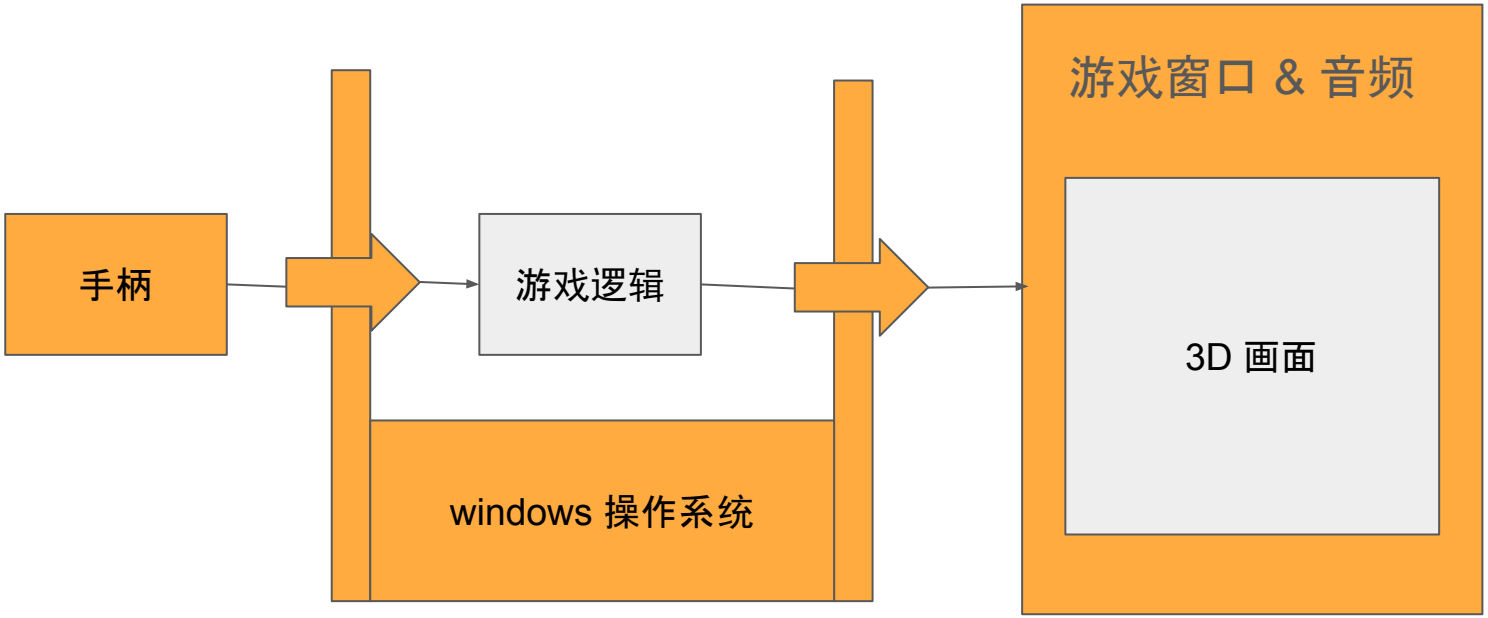
- cpu 指令集不同：手机的 cpu，比如高通骁龙 8gen3 是 arm64 的指令集，但是哈迪斯 2 是 x64 指令集的可执行文件
- 操作系统 api 不同：安卓提供的是修改后的 linux 内核，支持有限的 linux 内核的 syscall，哈迪斯 2是基于 windows 开发的，调用的 windows 操作系统 api
- 图形 api 不同：手机的 gpu 硬件是由高通的 gpu 驱动提供的 vulkan api来执行图形渲染的，但是哈迪斯 2 使用的是 windows 上的directx 12的 api 来渲染的

CPU 指令集



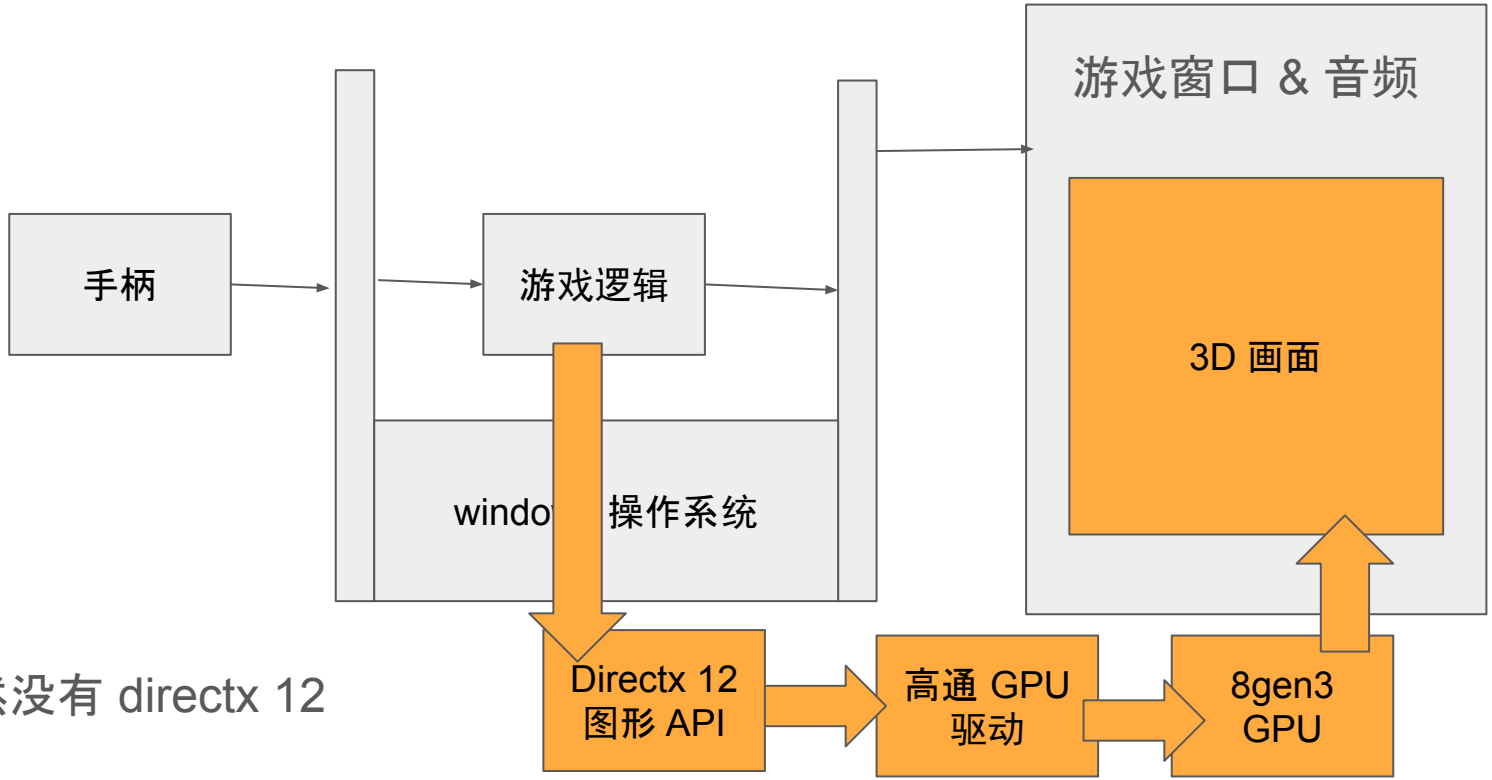
手机的 cpu，比如高通骁龙 8gen3 是 arm64 的指令集，但是哈迪斯 2 是 x64 指令集的可执行文件

操作系统 API - 接管输入输出



安卓提供的安卓的操作系统 api，哈迪斯 2是基于 windows 开发的，调用的 windows 操作系统 api

图形 API - 更直接控制 GPU



安卓上显然没有 directx 12

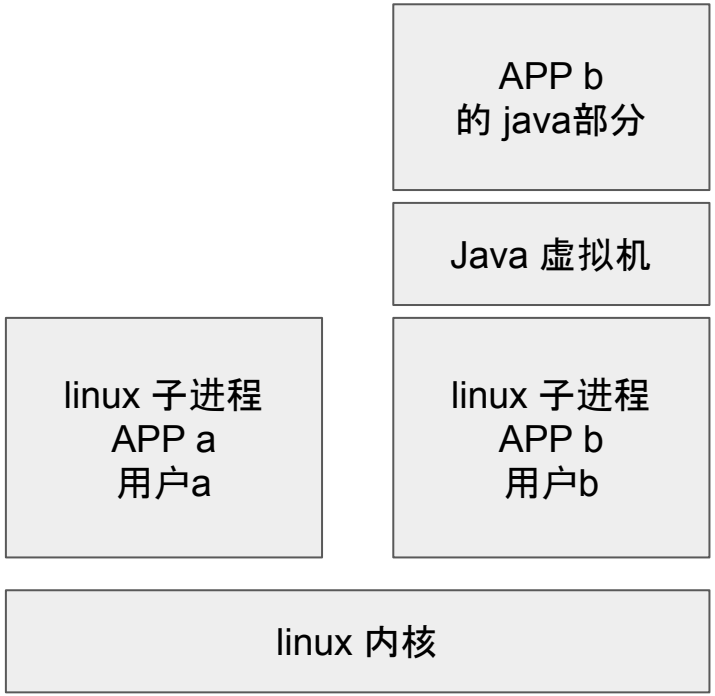
如何解决 cpu 指令集不同的问题？

- 用 arm64 cpu 执行“模拟CPU”软件，然后由 “模拟CPU” 来“执行”x64 指令编写的可执行文件（.exe），开源的 x64 CPU 模拟器：
  - qemu
  - box64
  - fex
  - blink
- winlator-proot 使用的是 box64
- winlator-glibc 使用的是 box64
- winlator-bionic 提供了 bionc 容器，使用的是 fex
- qemu 和 blink 的速度慢

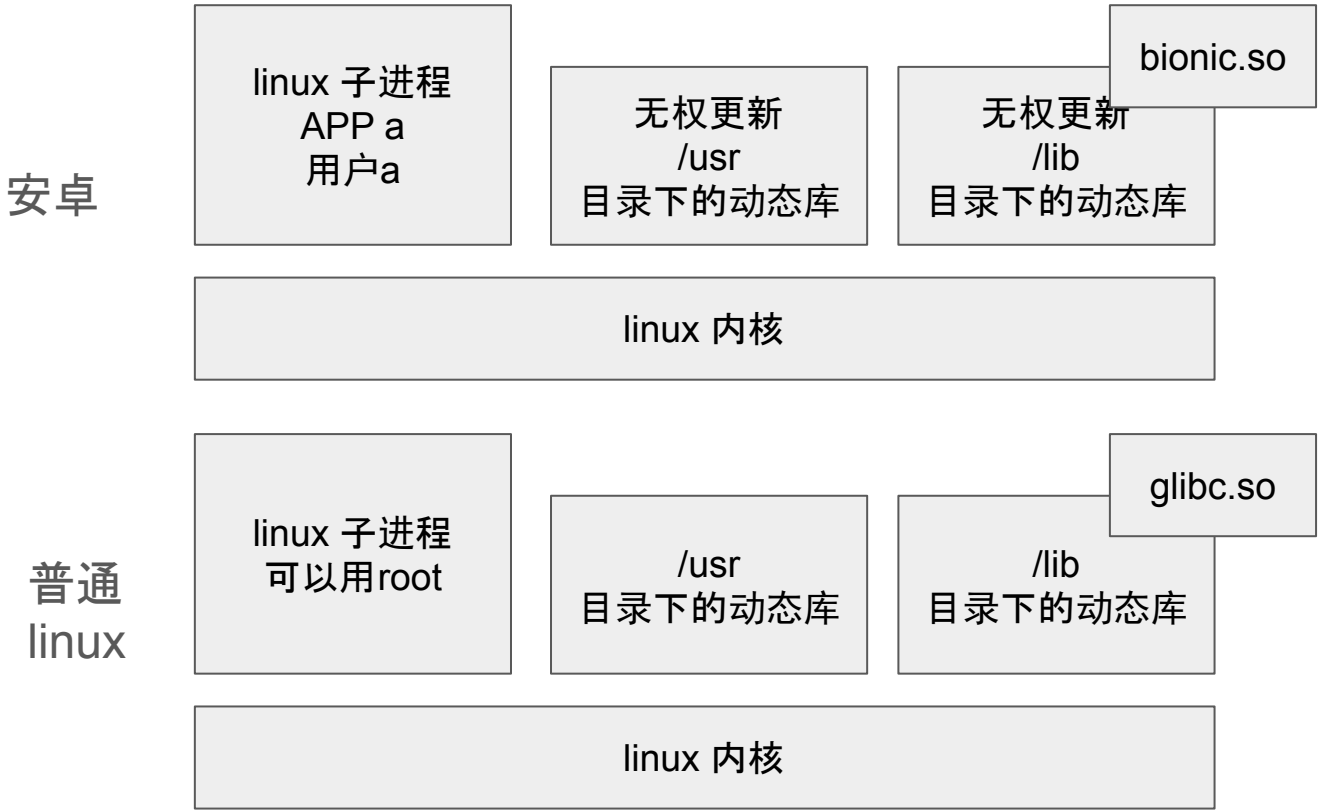
安卓启动 box64 进程，然后 box64 来跑游戏？

- 安卓 -> box64 -> 游戏的exe文件
- 类比：windows -> python解释器 -> *.py 文件
- 问题：
  - 安卓不是运行 apk 的么？
  - apk 不应该是 java 写的么？
  - 把 box64 用 java 重写变成一个 apk？

安卓其实是linux


APP b
的 java部分
Java 虚拟机
linux 子进程
APP a
用户a
linux 子进程
APP b
用户b
linux 内核

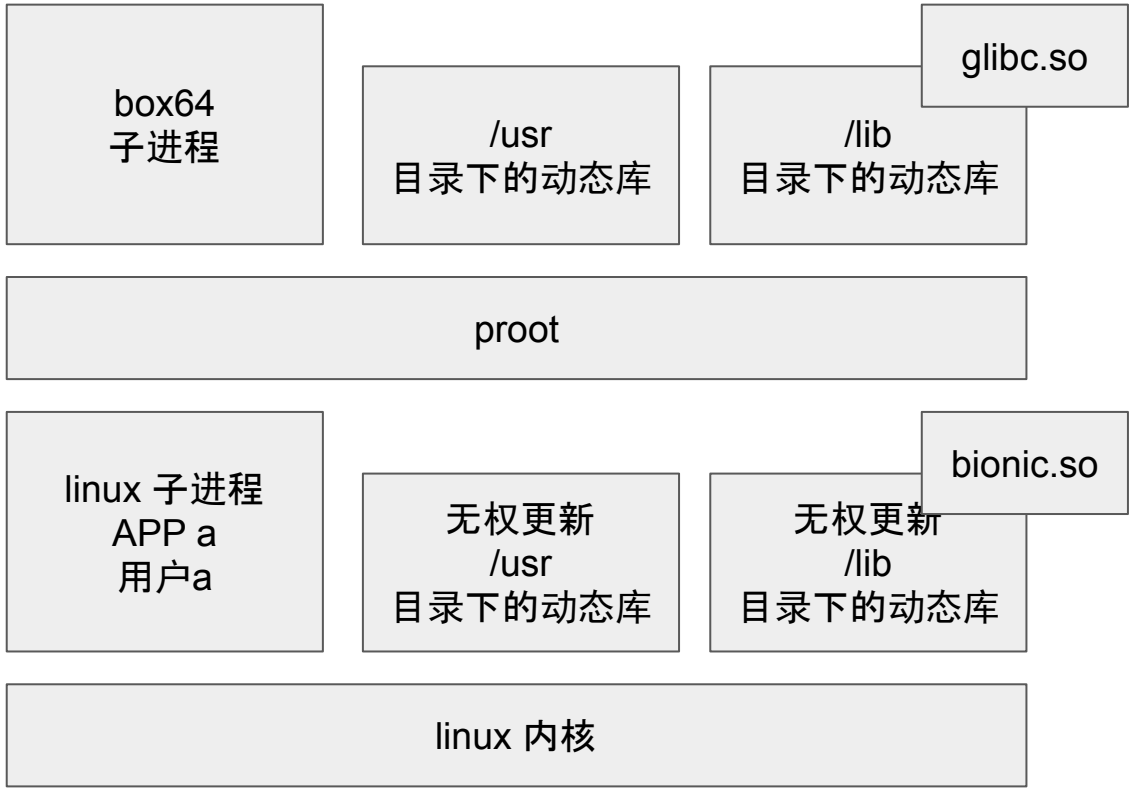
安卓并不是常规的 linux



安卓跑 linux 应用需要解决两个问题

- glibc 问题：linux 应用都使用了 glibc 这个基础库，安卓上没有 glibc，有一个类似的 bionic 库
- rootfs问题：linux 应用加载共享库需要从 /usr 或者 /lib 等目录下读取。但是安卓手机没有 root 权限，无法写入这这些目录

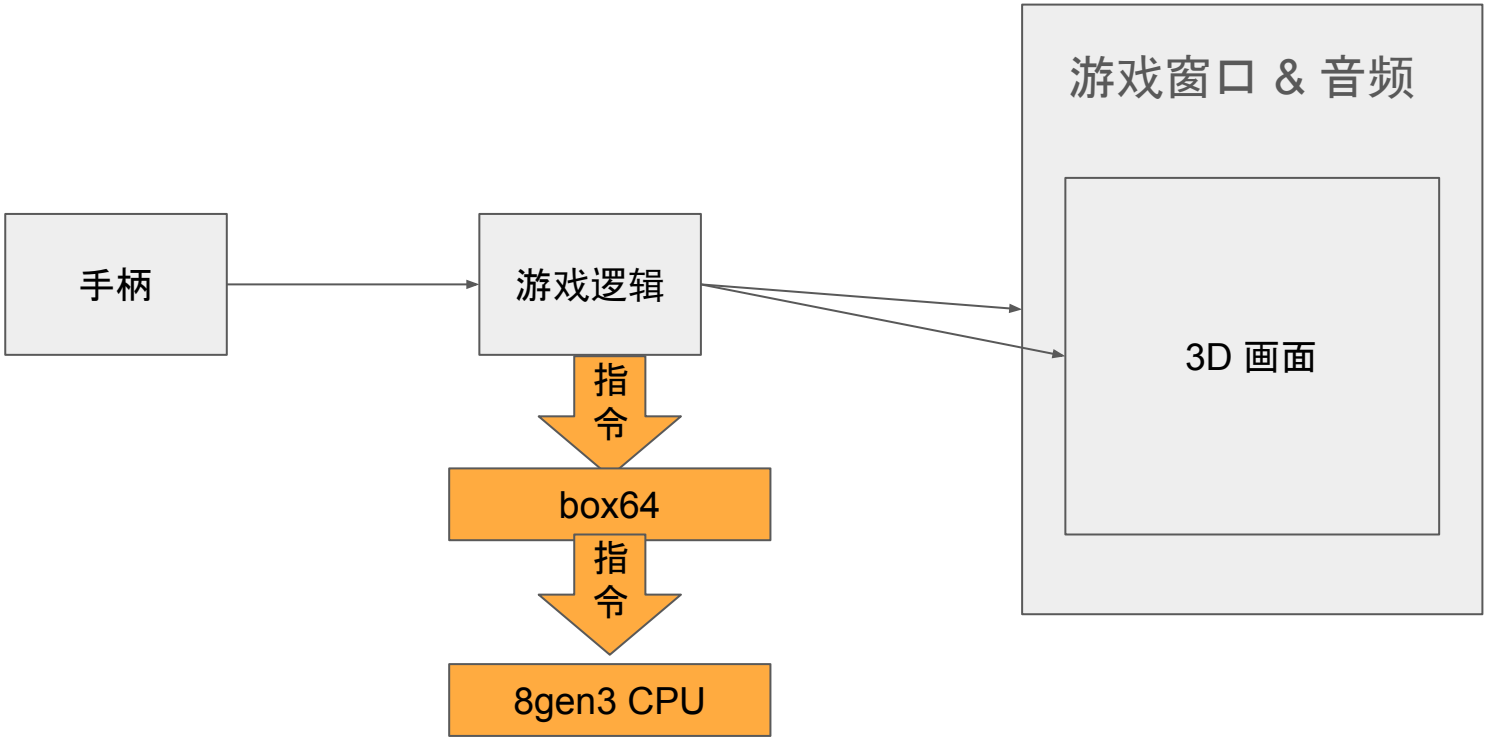
winlator-proot 是如何解决这些跑 box64 问题的？



winlator-proot 中 proot 的作用

- 解决 box64 依赖 glibc 的问题：无需改 box64，仍然提供 glibc
- 解决 rootfs 的问题：proot 用 ptrace 特权启动 box64，给 box64 进程虚构出了一个假的 rootfs，从而解决 /usr /lib 等目录无权写入的问题
- 启动代码：winlator/app/src/main/java/com/winlator/xenvironment/components/GuestProgramLauncherComponent.java at v7.1.0 · brunodev85/winlator
- 安卓 -> winlator进程 -> proot进程 -> box64进程 -> 游戏.exe？
- box64进程还无法直接启动 “游戏.exe”，中间还需要一层

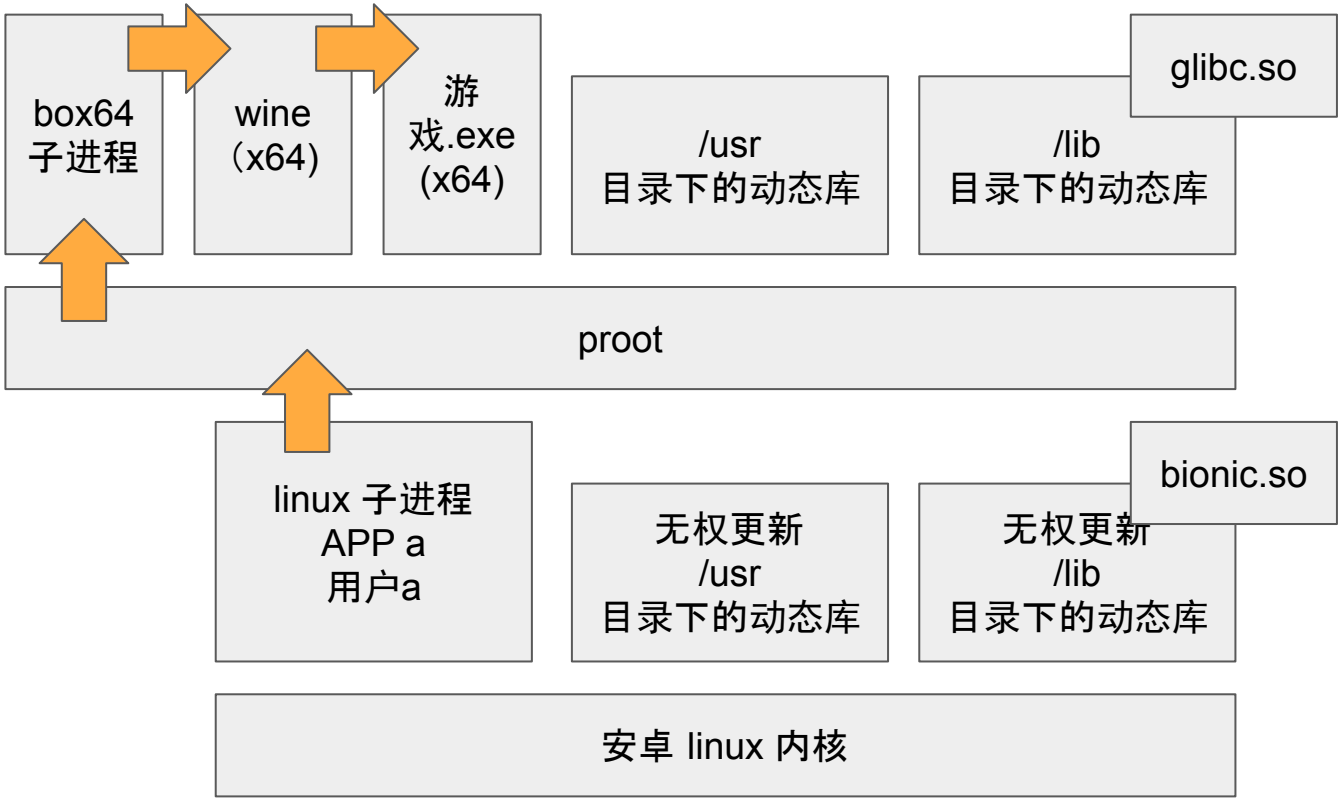
box64只解决了CPU指令集不同的问题



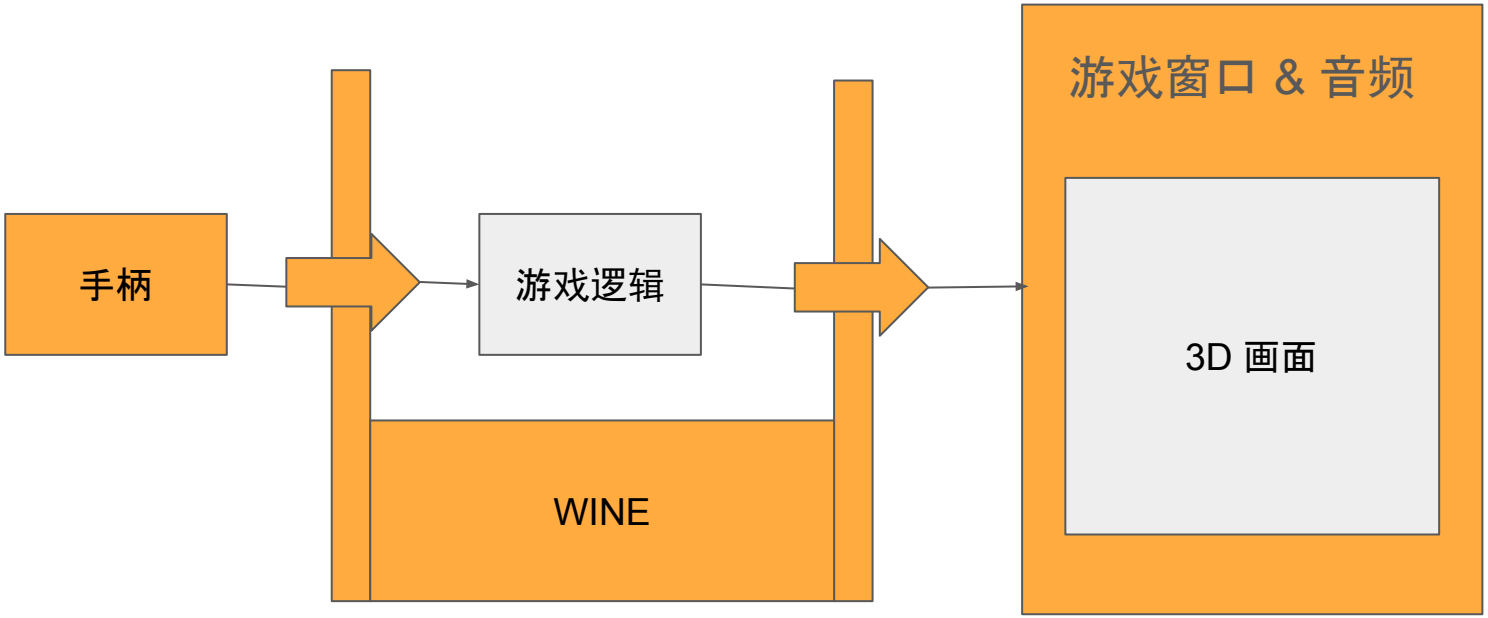
proot 中的 box64 跑游戏还需要解决两个问题

- 游戏.exe 依赖了 windows 的操作系统 api，但是 proot 提供的是一个 linux 环境，跑在安卓的linux内核上
  - 需要手柄输入
  - 需要windows窗口的图形输出
  - 需要音频输出
- 游戏.exe 依赖了 windows 的 directx 12 的图形api，但是 proot 环境下没有这个图形api，游戏无法直接操纵高通 8gen3 的 gpu 芯片

box64 -> wine -> 游戏.exe



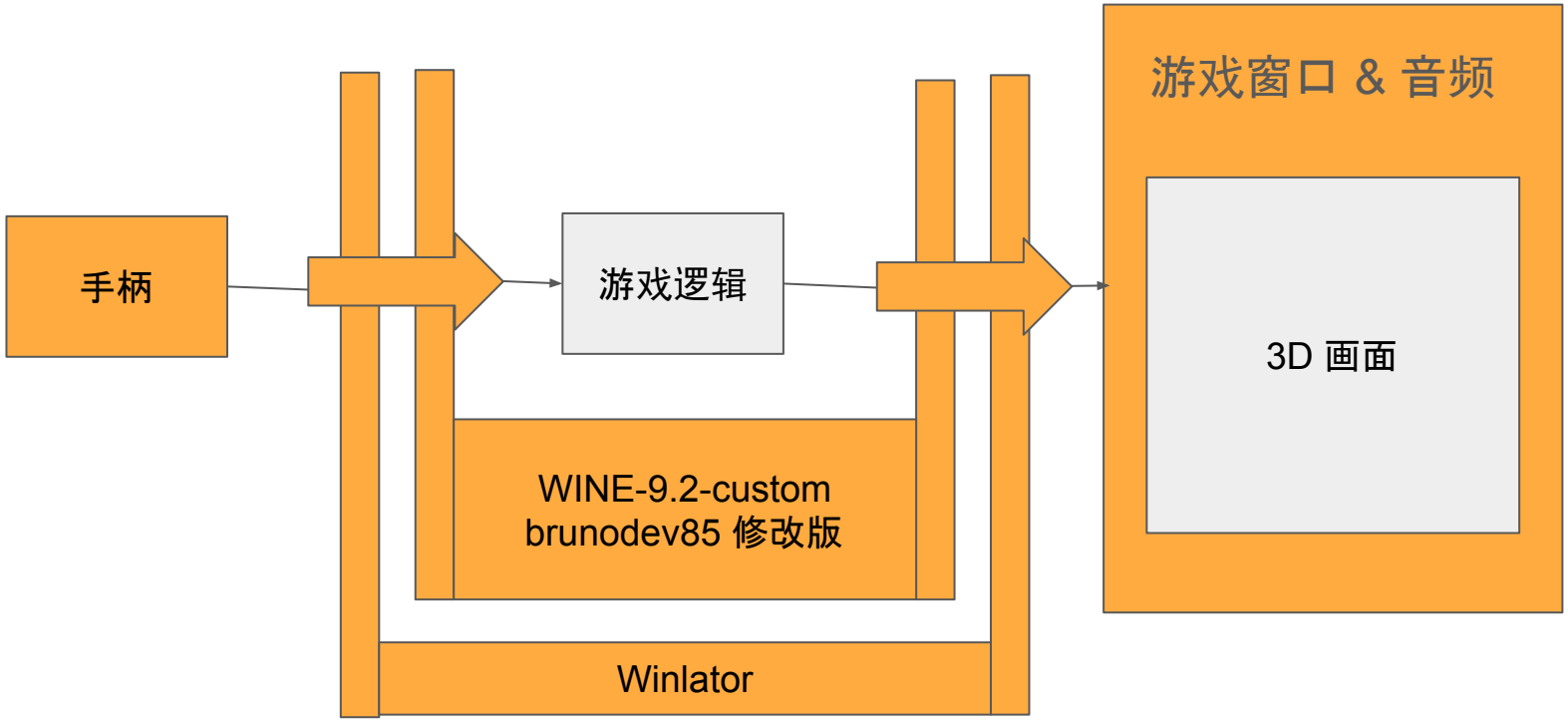
WINE 来提供 windows 操作系统的 api



WINE 如何能控制安卓的？

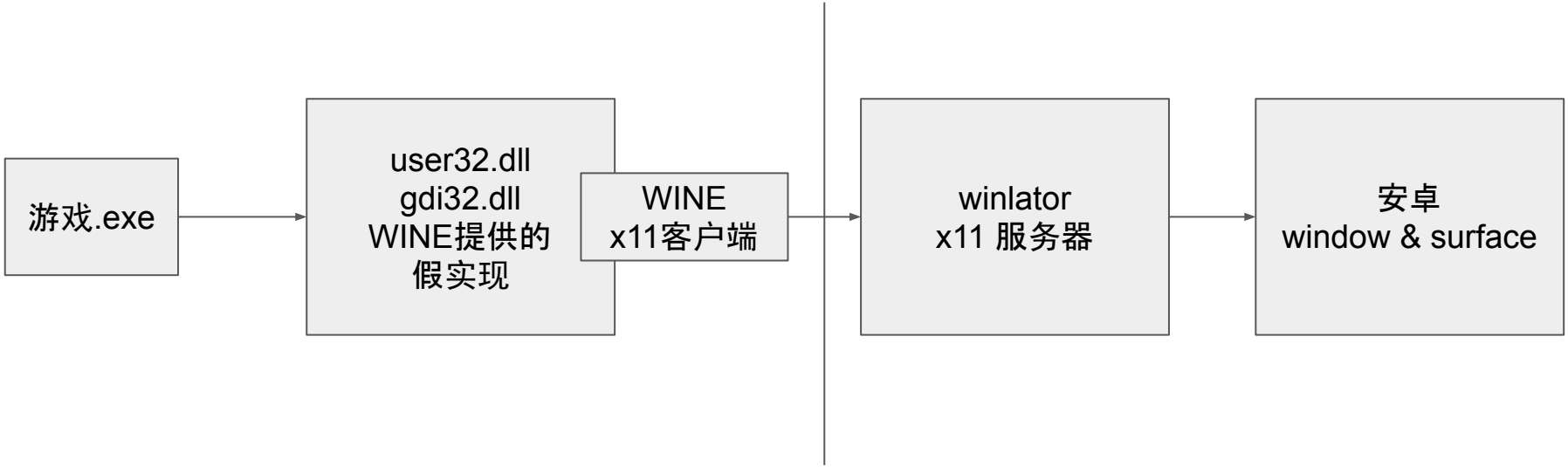
- WINE 是一个跑在 proot 的 linux 环境之内的，proot 只是把 linux 内核的 syscall 暴露给了其上的环境，并没有提供安卓 API 的访问能力
- 那 WINE 如何能操控安卓呢？
  - 手柄的输入怎么从安卓到 WINE 呢？
  - WINE 的 windows 窗口怎么绘制在安卓手机的屏幕上呢？
  - WINE 的音频输出怎么从安卓手机的喇叭 输出呢？
- winlator 修改了 WINE 的源代码，让输入输出走 winlator 的渠道

Winlator 桥接了安卓和WINE：窗口



[brunodev85/wine-9.2-custom: Main Wine version used in Winlator]

窗口绘制过程

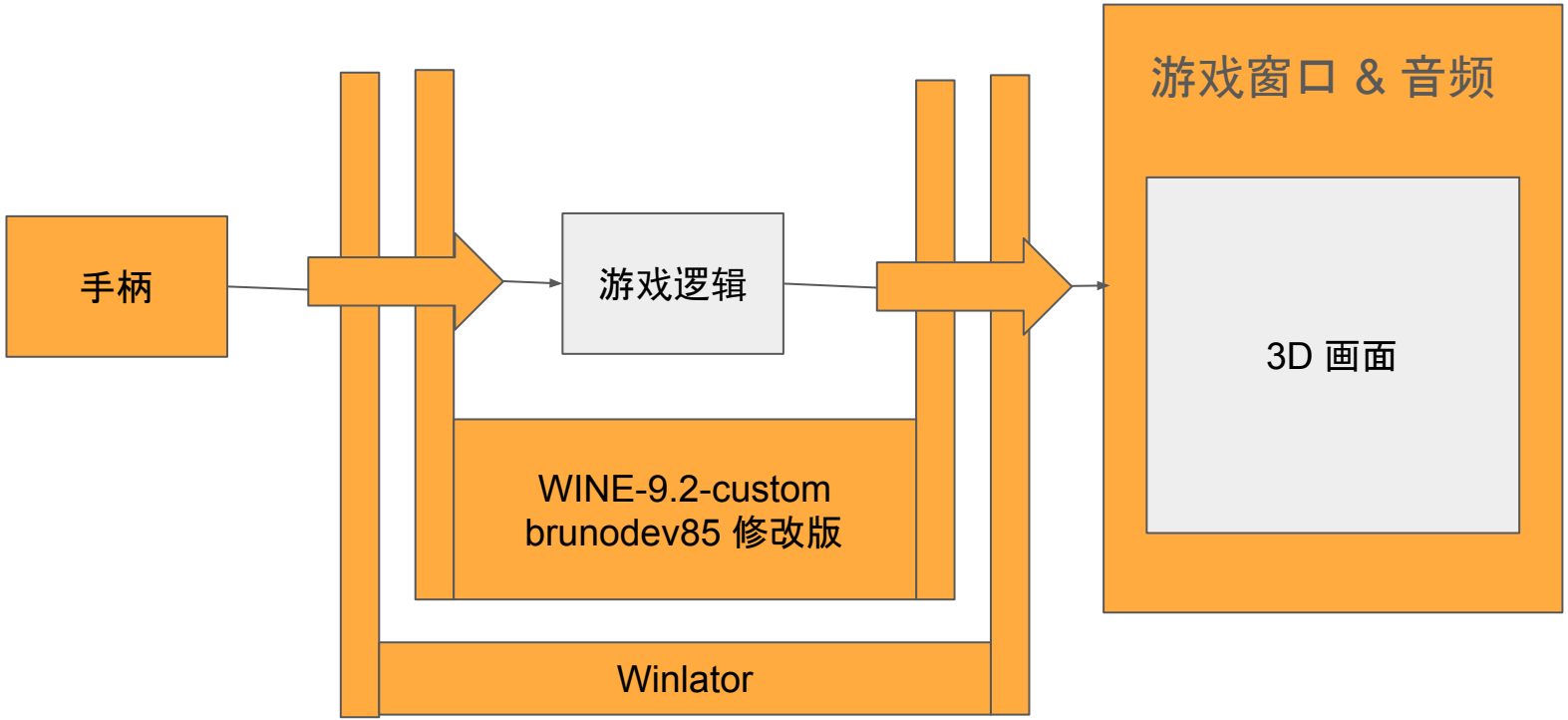


envVars.put("DISPLAY", ":0");
对应 unix domain socket

termux 开源项目和 winlator 的关系

- termux 提供了大量linux包的安卓移植：其中就有 proot，使得 proot 链接安卓的 bionic 而不是 glibc
- termux 提供了完整的 x11 server
- winlator 没有直接使用 termux的源代码或者构建产物
- winlator 内置了 proot 的 c++ 代码，用 ndk 编译到自己的 apk 里
  - winlator/app/src/main/cpp/proot/CMakeLists.txt at v7.1.0 · brunodev85/winlator
- winlator 内置了 java 实现的简化版 x11 server，直接用 java 启动的 x11 unix domain socket端口
  - winlator/app/src/main/java/com/winlator/xserver at v7.1.0 · brunodev85/winlator

Winlator 桥接了安卓和WINE：音频



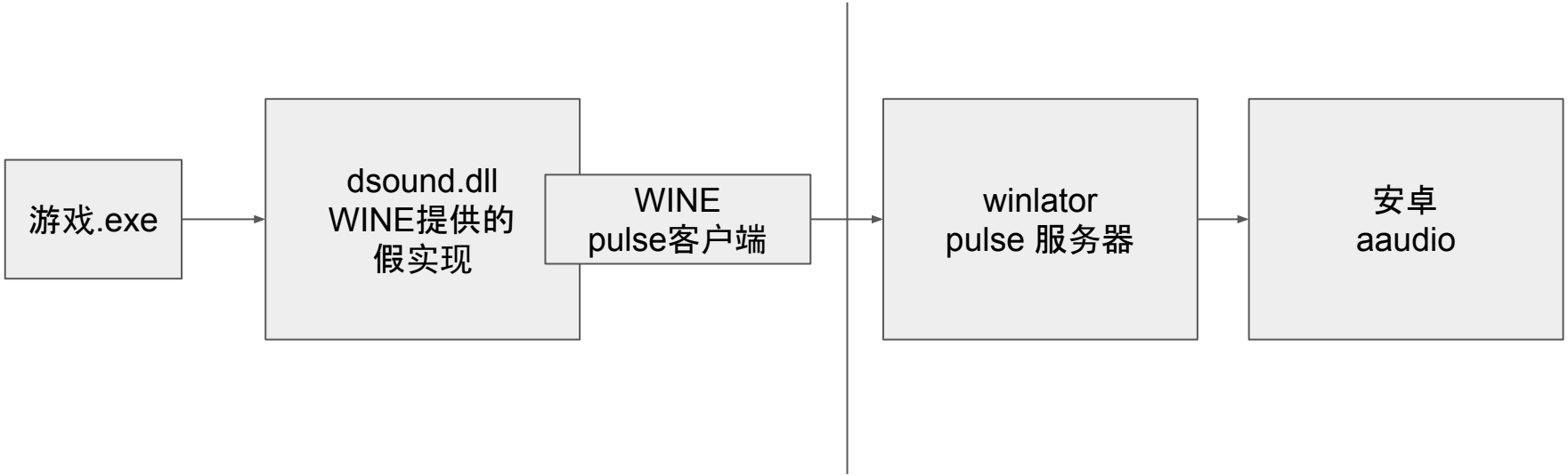
brunodev85/wine-9.2-custom: Main Wine version used in Winlator

WINE 输出音频的两个渠道

通过写入 windows 注册表，选择 Audio 输出是 alsa 还是 pulse audio

```
try (WineRegistryEditor registryEditor = new WineRegistryEditor(userRegFile)) {
            if (audioDriver.equals("alsa")) {
                registryEditor.setStringValue("Software\\Wine\\Drivers", "Audio", "alsa");
            }
            else if (audioDriver.equals("pulseaudio")) {
                registryEditor.setStringValue("Software\\Wine\\Drivers", "Audio", "pulse");
            }
        }
```

WINE 走 pulse audio 渠道输出声音

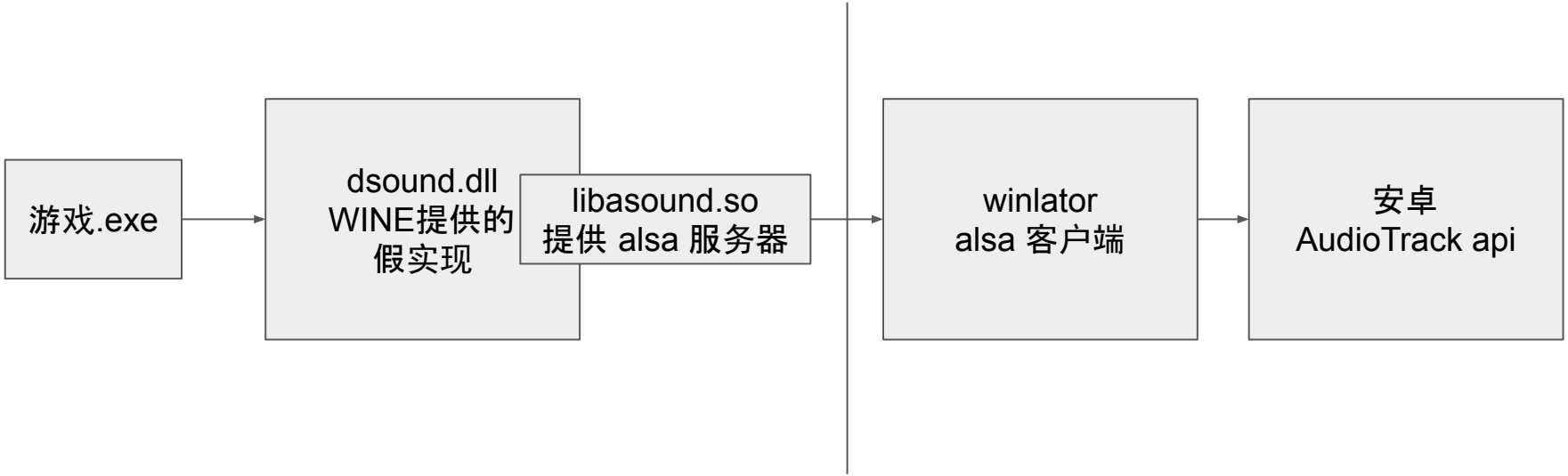


envVars.put("PULSE_SERVER",
UnixSocketConfig.PULSE_SERVER_PATH);
对应 unix domain socket

winlator 的 pulse audio server

- winlator/app/src/main/jniLibs/arm64-v8a/libpulseaudio.so at v7.1.0 · brunodev85/winlator
- 源代码：brunodev85/pulseaudio-android: PULSEAUDIO SOUND SERVER
- pulse audio client 是 linux 的标准组件，被动态链接到 wine 里了。直接支持 PULSE_SERVER 这个环境变量
- 输入是 unix domain socket 连接到 wine，pulse 协议
- 输出是安卓的 aaudio

WINE 走 alsa 渠道输出声音

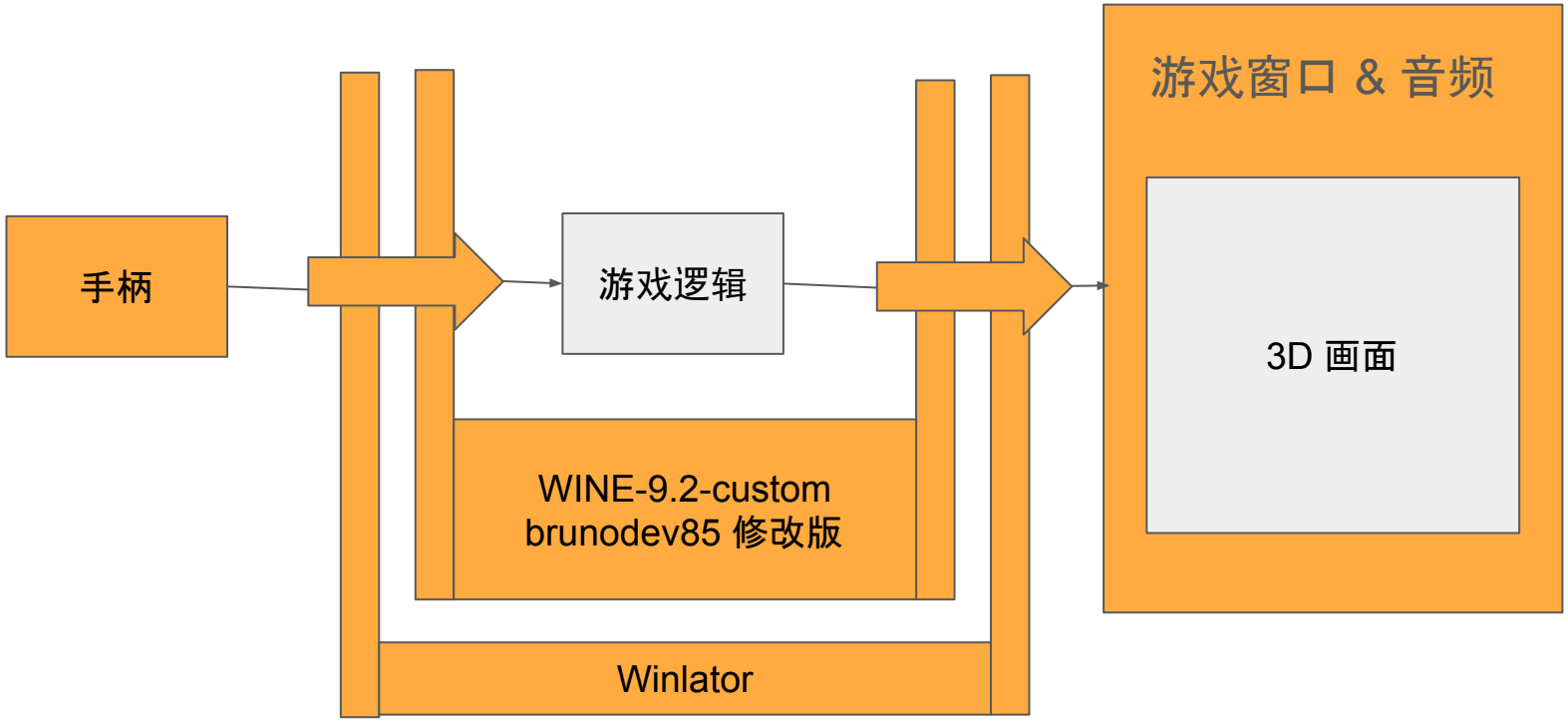


envVars.put("ANDROID_ALSA_SERVER",
UnixSocketConfig.ALSA_SERVER_PATH);
对应 unix domain socket

winlator 的 alsa 客户端和服务器

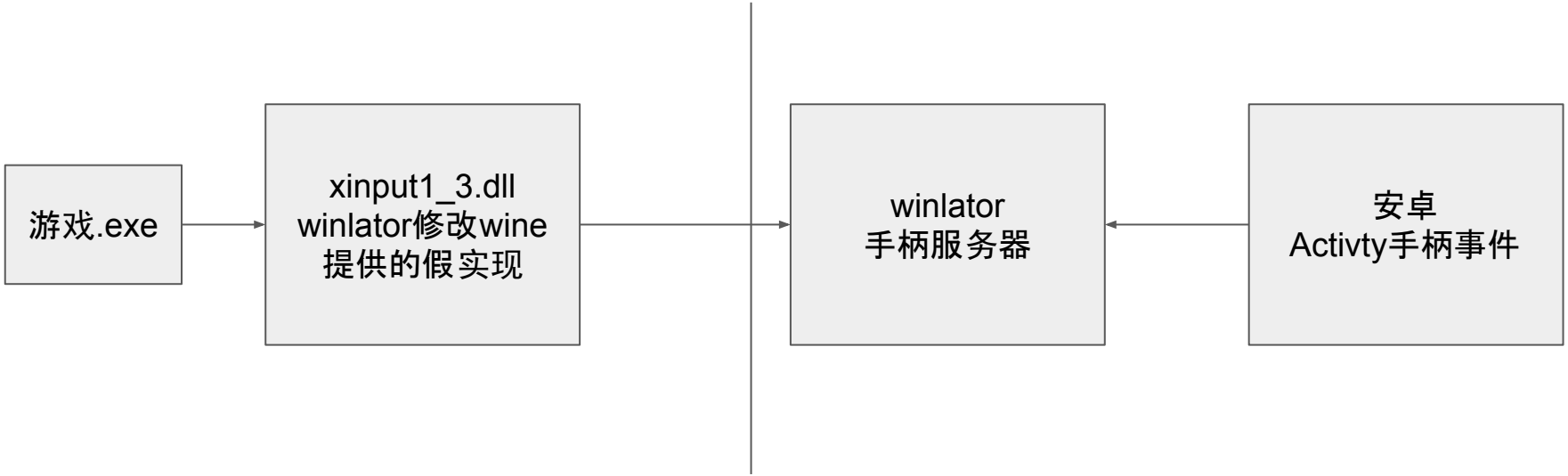
- 正牌的 alsa 是没有客户端服务器的，是走内核syscall调用内核驱动的
- wine 访问 libasound.so，实际上是 winlator 提供的假实现 winlator/audio_plugin at v7.1.0 · brunodev85/winlator 暴露 alsa server
- 然后通过 unix domain socket 连接，winlator 用客户端连上这个 alsa server winlator/app/src/main/java/com/winlator/alsaserver/ALSAClient.java at v7.1.0 · brunodev85/winlator
- 后续 winlator 8 把 alsa 客户端的一部分改成用 c++ 来写，提高稳定性

Winlator 桥接了安卓和WINE：手柄

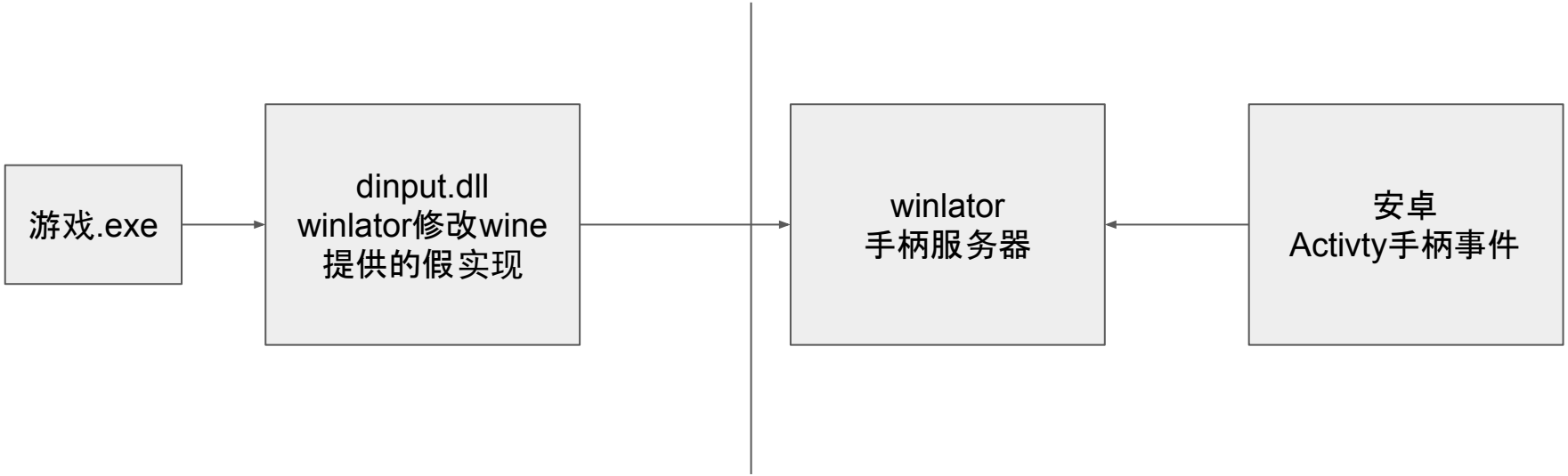


[brunodev85/wine-9.2-custom: Main Wine version used in Winlator]

WINE 走 xinput 获取手柄状态



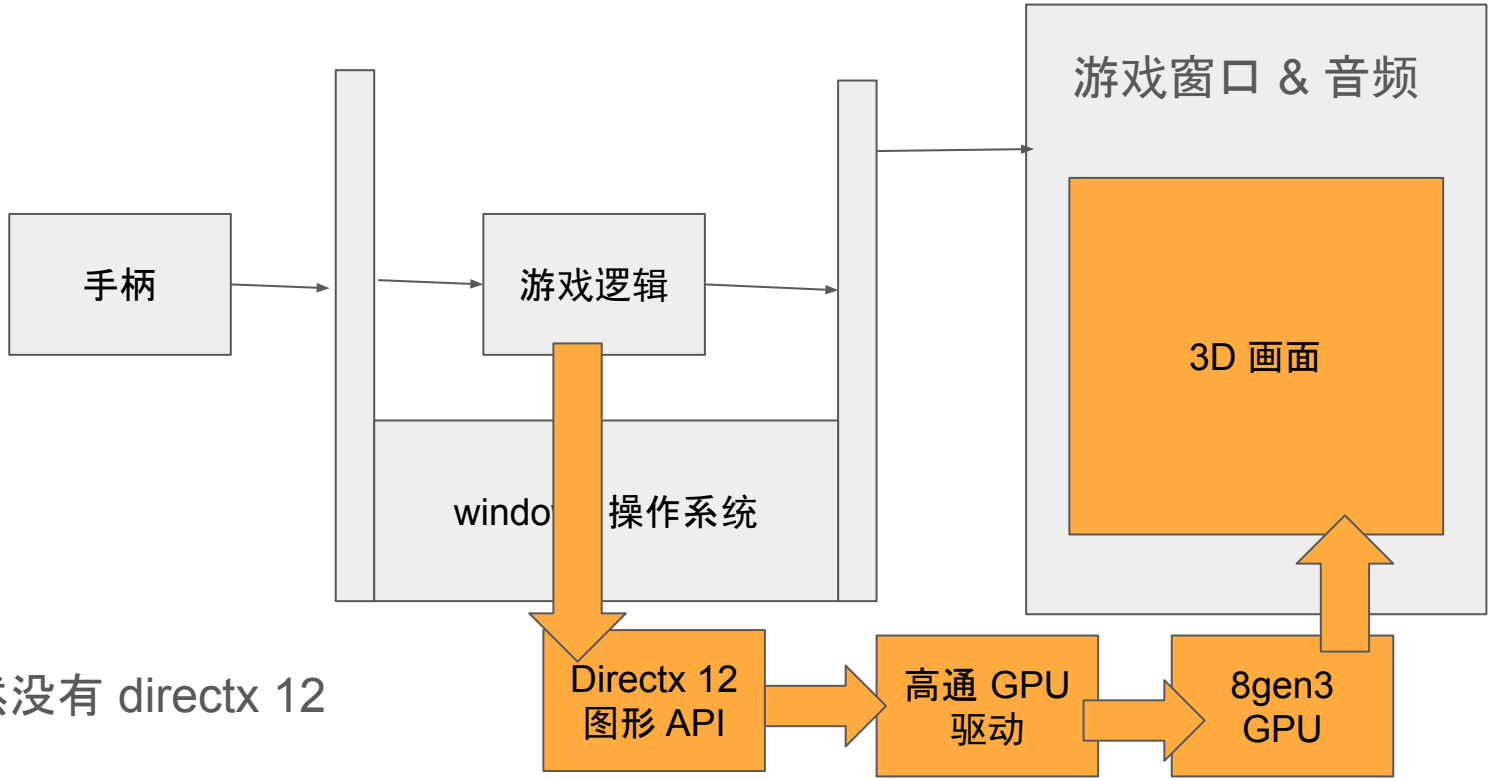
WINE 走 dinput 获取手柄状态



winlator 的手柄客户端和服务器

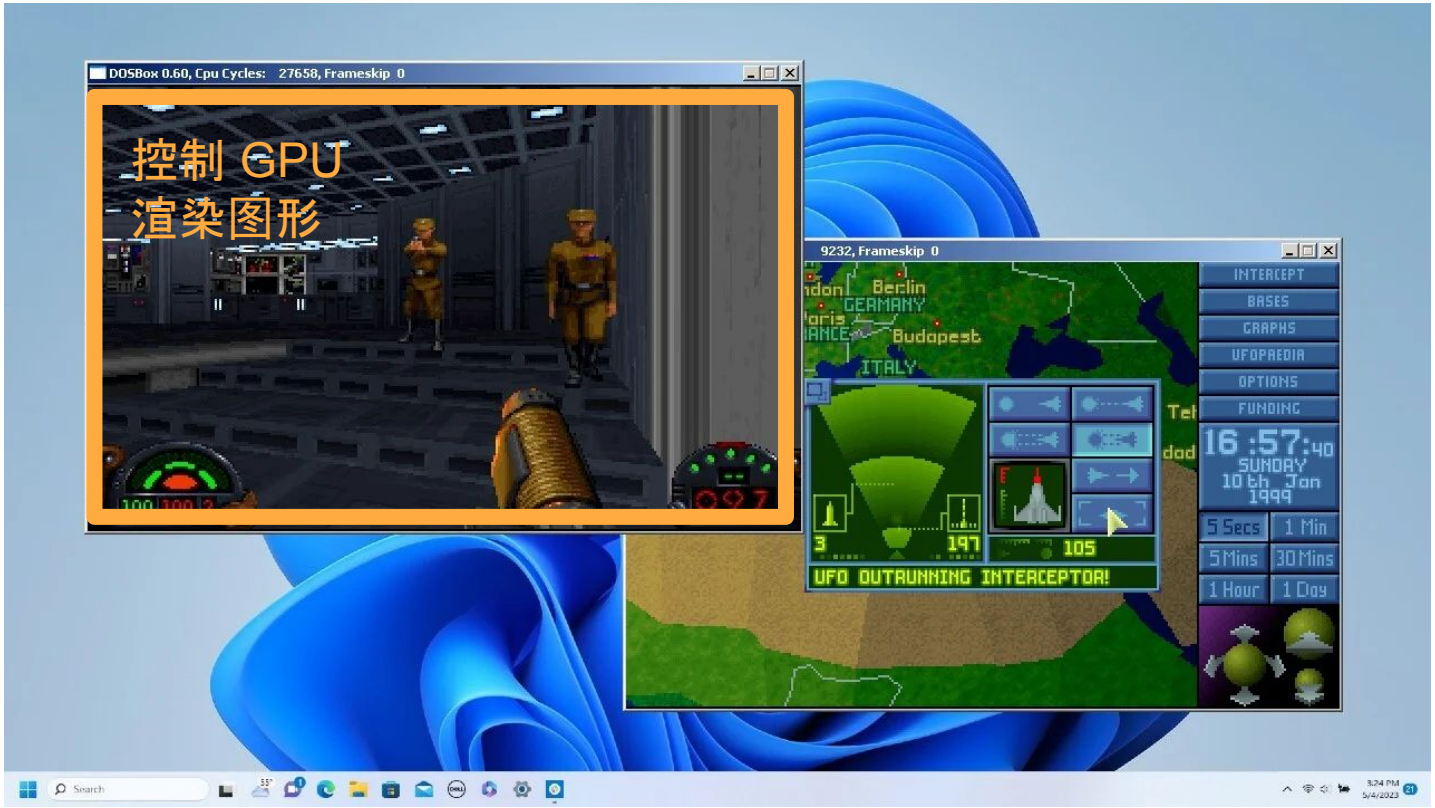
- 手柄客户端部分在 winlator 的 wine 修改代码里 https://github.com/brunodev85/wine-9.2-custom
- 手柄服务器在 winlator 的代码里 winlator/app/src/main/java/com/winlator/winhandler/WinHandler.java at v7.1.0 · brunodev85/winlator
- 无论安卓是怎么连接到手柄的，有线或者无线，最后都是通过 activity 的事件分发给 winlator 的。安卓怎么驱动手柄，那是安卓的事情
- winlator 拿到安卓给的事件，更新手柄的当前信息到内存中
- 无论 dinput 还是 xinput 最终都是访问 winlator 获取手柄的当前信息
- 手柄不被游戏识别，和安卓怎么连接手柄是无关的，和 winlator 修改的 wine xinput 和 dinput 有关

游戏怎么控制 GPU 绘制 3D 画面？

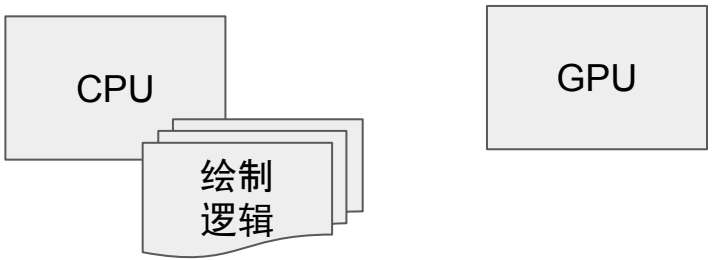


安卓上显然没有 directx 12

游戏直接控制 GPU 渲染图形

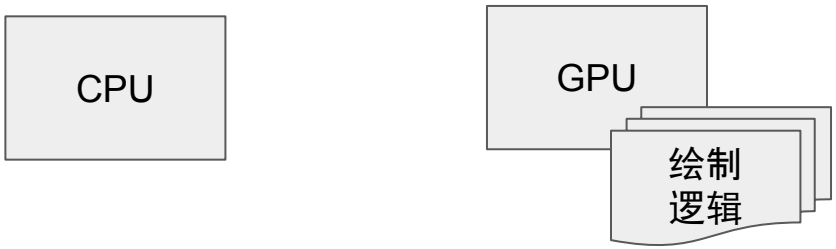


游戏跑在 CPU 上，控制 GPU 需要三步：1 准备绘制逻辑



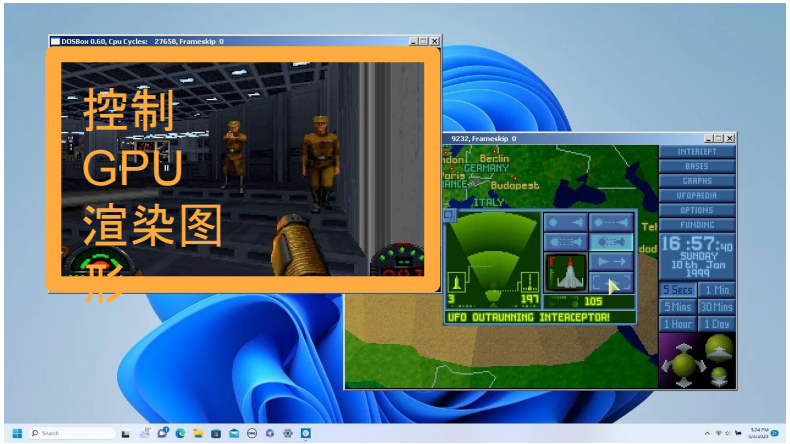
游戏准备好绘制逻辑，也就是给 GPU 的程序
这部分就是 directx 11, directx 12, opengl, vulkan 这些 API 准备的东西

游戏跑在 CPU 上，控制 GPU 需要三步：2 驱动 GPU 渲染



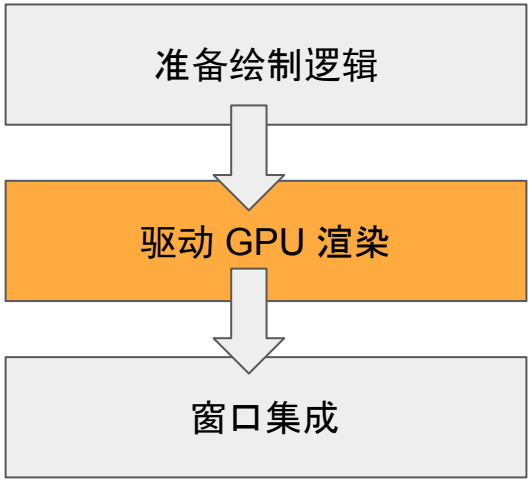
CPU 把绘制逻辑交给 GPU 执行，变成 GPU 内部的指令进行渲染执行，获得图片

游戏跑在 CPU 上，控制 GPU 需要三步：3 窗口集成



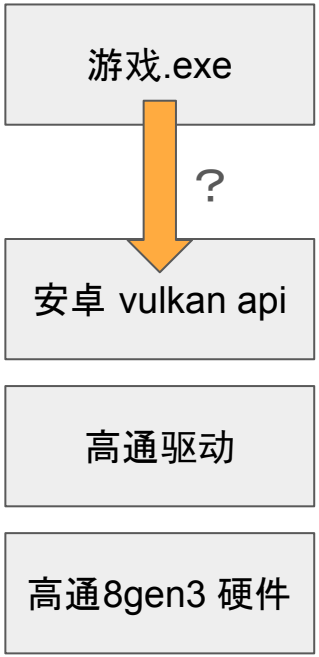
给 GPU 一个空白的画布进行绘制，绘制完之后把画布上的内容和窗口的其他部分进行拼接，变成最终的画面。

游戏怎么控制 GPU 绘制 3D 画面？需要解决 3 个子问题

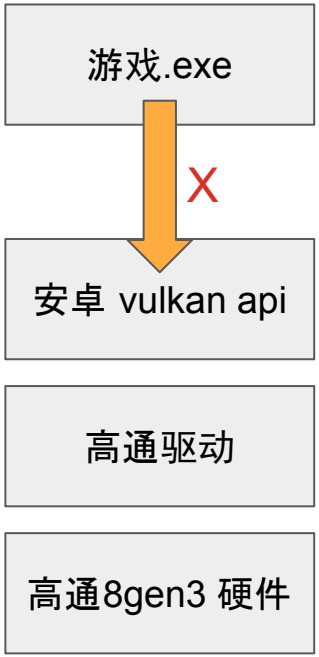


驱动 GPU 是最大的难点

游戏.exe 为什么不能直接调用安卓 api？



游戏.exe 为什么不能直接调用安卓 api？



游戏.exe

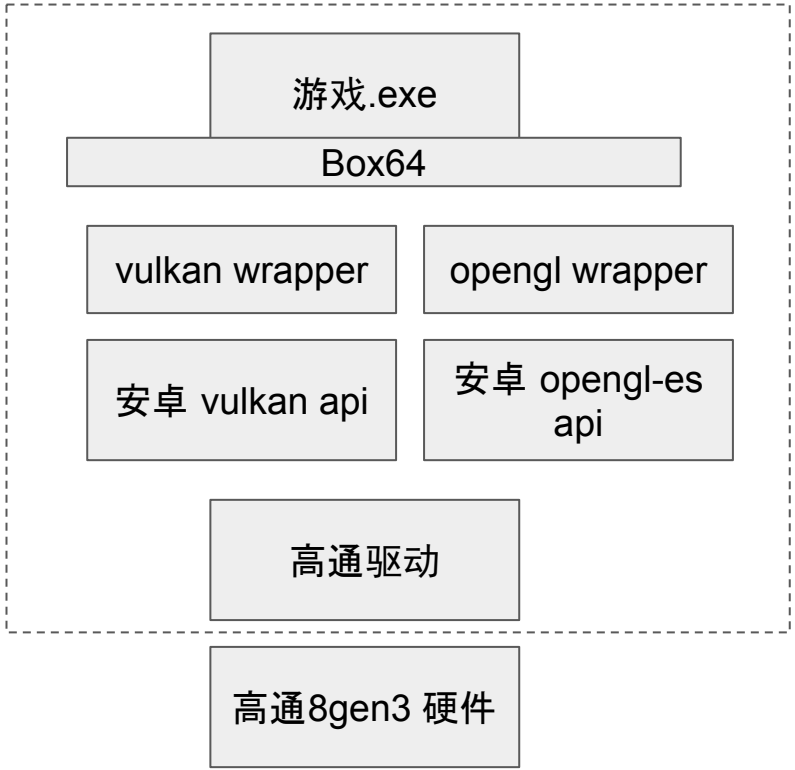
- windows pe 格式
- 依赖 windows 的 dll
- x64 指令集

/system/lib64/libvulkan.so

- linux elf 格式
- 依赖 bionic 这个 libc 基础库
- arm64 指令集

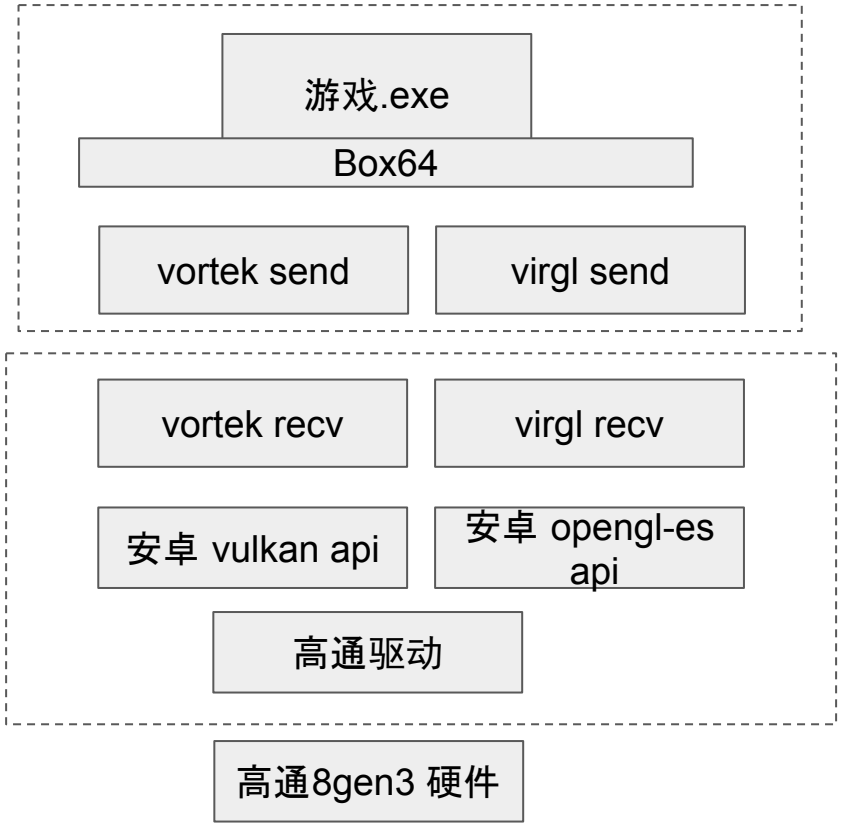
完全不可能直接在“游戏.exe”的进程内直接加载安卓的 vulkan 库

游戏.exe 访问到 GPU 的三种途径：1 直接链接安卓库



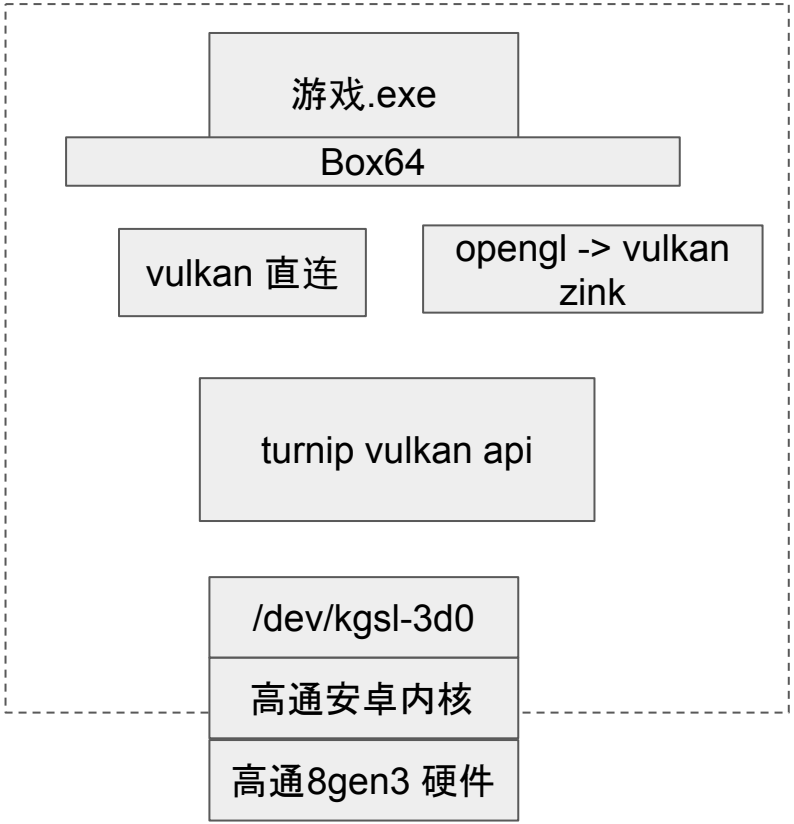
- 整个虚线部分是一个进程
- Box64 允许 x64 的进程访问 arm64 的动态依赖库
- wrapper 驱动在依赖库函数层面做了 api 直接转发
- box64 需要是 bionic libc 的
- vulkan / opengl wrapper 也必须是 bionic libc 的
- 因为安卓的 vulkan / opengl 库是 bionic libc 的
- 一个进程内不能混用 bionic 和 glibc 这两套 libc

游戏.exe 访问到 GPU 的三种途径：2 网络转发



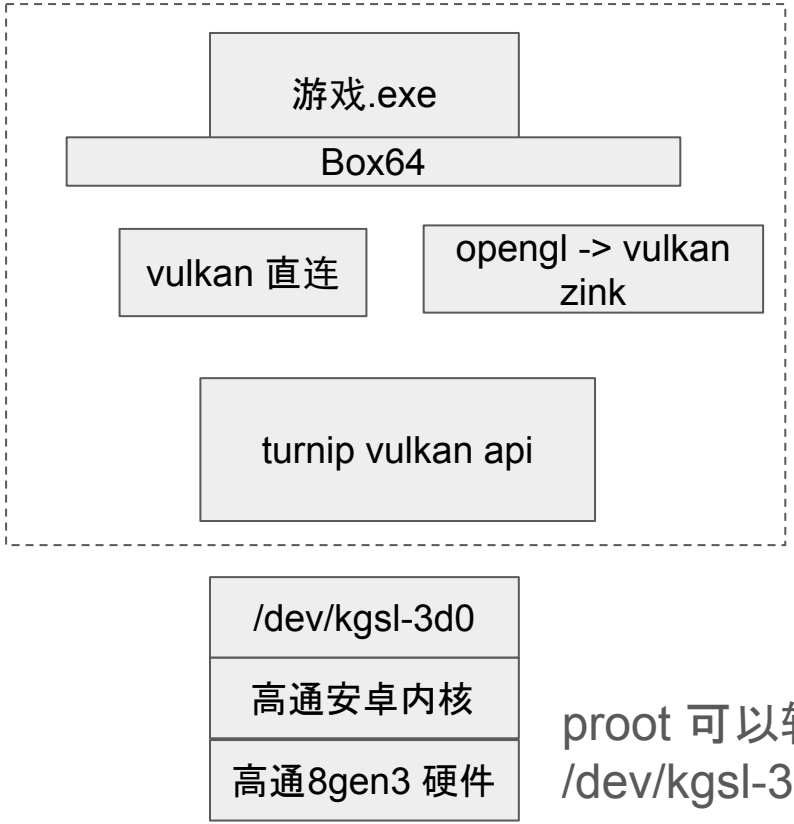
- 网络转发是在两个独立进程之间的，需要序列化和反序列化
- 通用性好，发送端可以是 glibc 的，接收端可以是 bionic 的。发送端可以是在 proot 容器之内
- 性能差，比直接进程内链接多了序列化，传输，和反序列化的开销
- winlator 的 vulkan 转发是 winlator 9 引入的 vortek 驱动，没有开源
- linux 社区有类似的 vulkan 转发 virtio venus
- winlator 的 opengl 转发是 virgl，是从 linux 社区移植过来的开源组件

游戏.exe 访问到 GPU 的三种途径：3 直接接管硬件

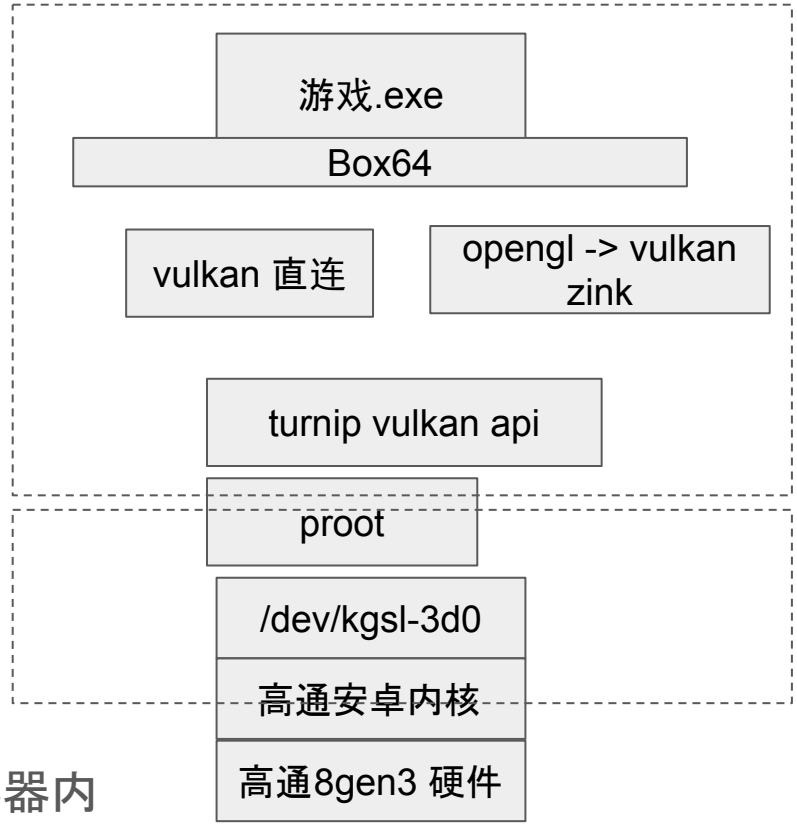


- 高通的 GPU 直接在安卓内核里提供了 /dev/kgsl-3d0 这个文件路径
- 通过这个文件可以直接控制高通 GPU 硬件
- 开源的 turnip 驱动代替了高通私有驱动，直接提供了 vulkan api
- zink 把 opengl 转换成 vulkan

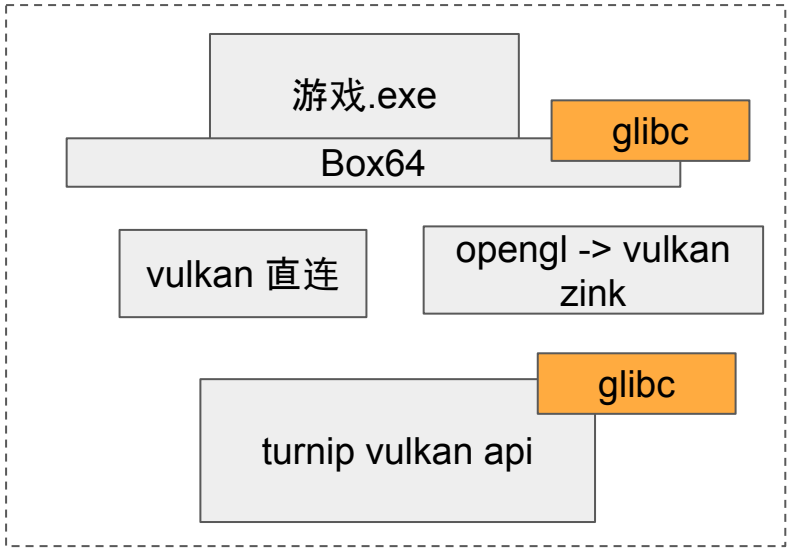
turnip 在 proot 内和 proot 外的区别

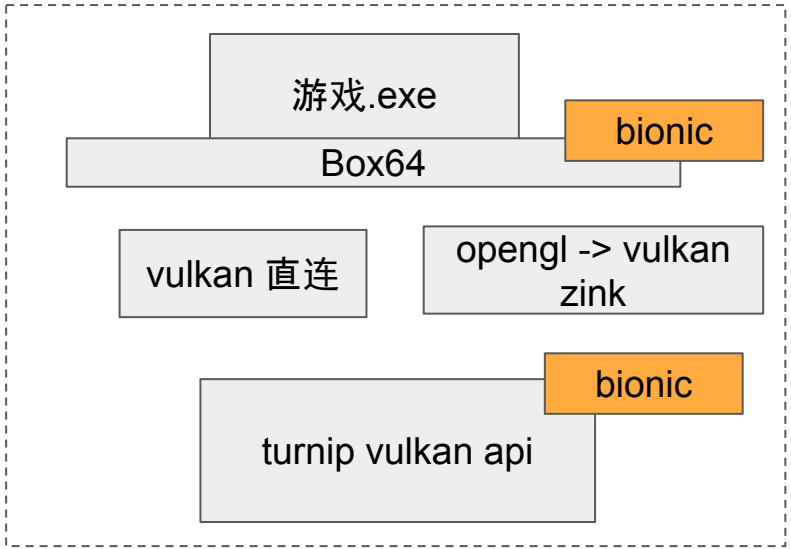


proot 可以转发
/dev/kgsl-3d0 到容器内



turnip 用 glibc 和 bionic 的区别



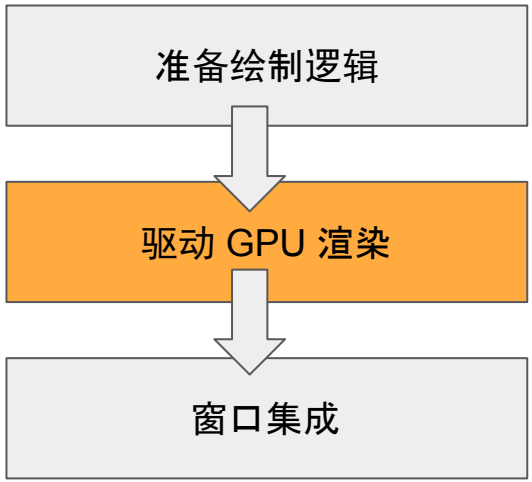


取决于 box64
是怎么编译的

turnip 为什么要分 glibc 和 bionic 的

- 从 https://github.com/K11MCH1/AdrenoToolsDrivers 下载的是 bionic 的，很多安卓上的模拟器都能用，因为这些模拟器都是 bionic 的
- 从 K11MCH1/WinlatorTurnipDrivers: A repository for Winlator Mesa Turnip drivers. (github.com) 下载的是 glibc 的，只给 winlator 用
- 使用 glibc 的理由
  - 如果使用了 proot 跑 ubuntu，ubuntu 的文件都是 glibc 的
  - box64 用 bionic 编译没有官方直接支持，编译出来稳定性和性能都有问题。盖世游戏有自己的 patch 来解决这个问题，但是没有开源
- 使用 bionic 的理由：更快
- 同一个进程只能选择一个 libc 动态链接库，无法混用 glibc 和 bionic

游戏.exe 显卡驱动的选择

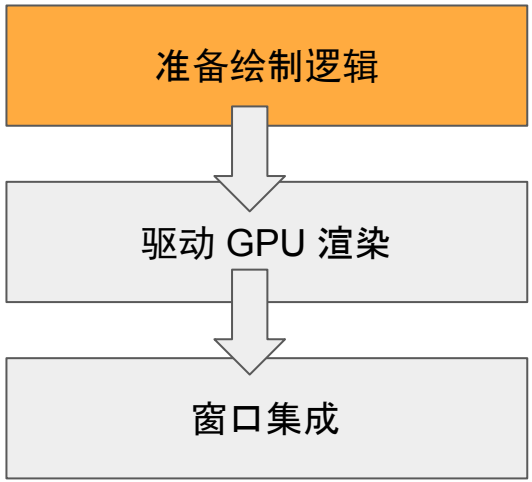


1、wrapper：性能最好，只能使用安卓操作系统的 GPU 驱动版本，如果这个驱动的API 实现质量不好，就会有图形错误

2、网络转发：virgl 或者 vortek。性能差一些，最终使用的驱动和 wrapper 是一样的，但是可以在 proot+glibc 或者直接 glibc 的环境下工作

3、turnip：性能和 wrapper 各有胜负。能够选择 turnip 的版本，可以不依赖安卓系统升级去独立升级 GPU 驱动，因为驱动写得好所以图形错误更少

游戏.exe 准备绘制逻辑需要解决的问题



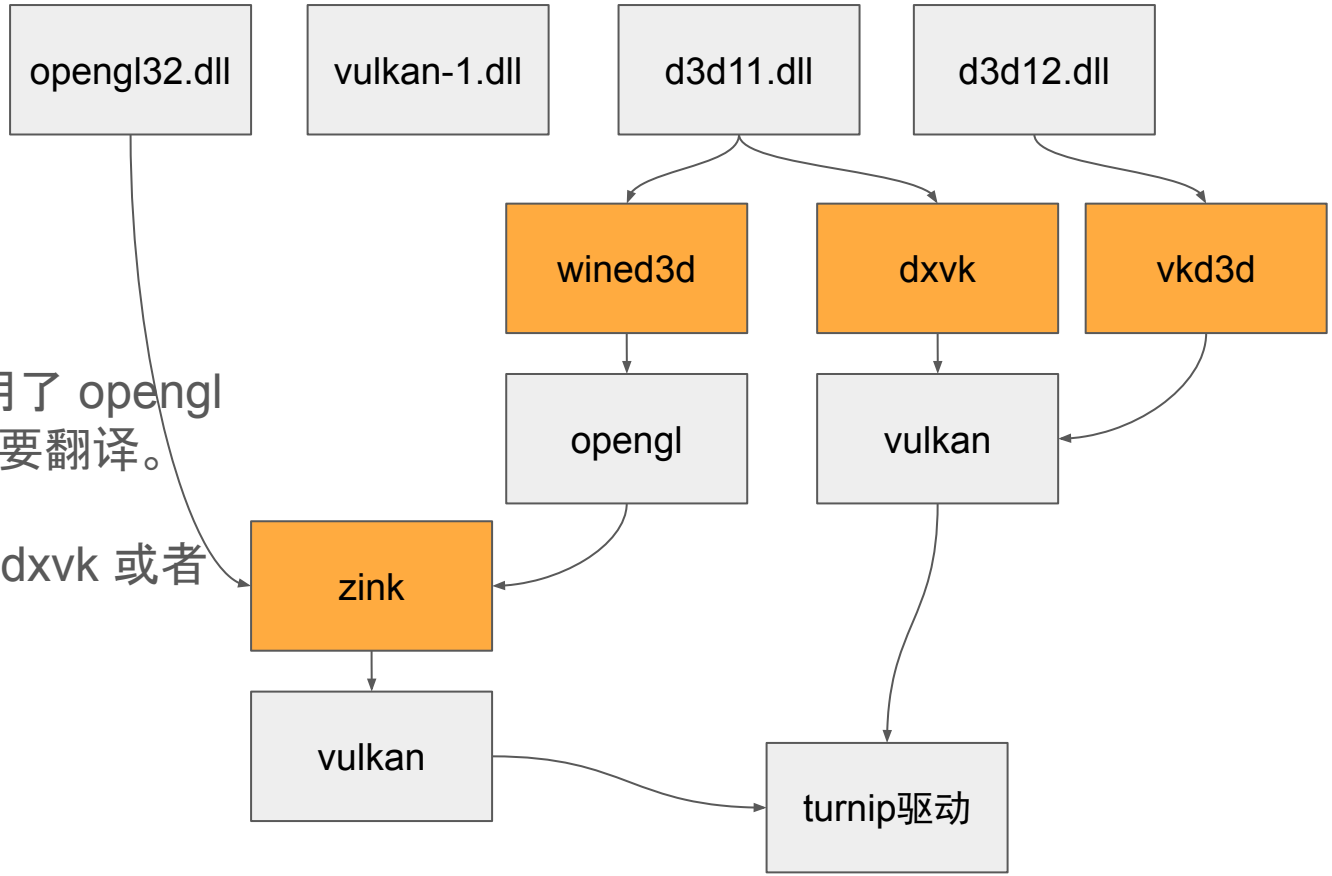
windows 游戏可能使用的图形 api 包括

- opengl
- vulkan
- directx
  - directx 11 以及以下
  - directx 12

无论游戏使用哪个图形 API 来准备绘制逻辑，都要能支持

翻译图形 API

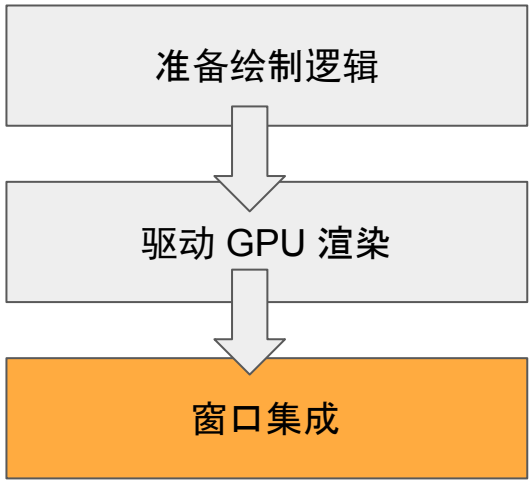
- 如果游戏.exe直接用了 opengl 或者 vulkan 则不需要翻译。wine 会直接转发
- directx11 和以下用 dxvk 或者 wined3d
- directx12 用 vkd3d



选择如何翻译 API

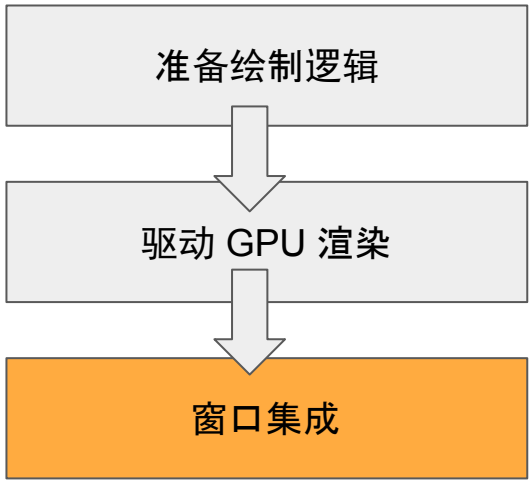
- wined3d 虽然比 dxvk 慢，但是对老游戏，或者 2d 游戏来说可能也足够快了
- box64 有一个特殊的 box32 模式，只支持 wined3d
- 直接使用系统驱动（社区俗称 prop driver，也就是 proprietary driver 私有驱动的简称）会有一些图形错误的可能。dxvk 的新版本为 8elite 的 prop driver 做了一些问题规避
- 另外一个社区解决图形错误的努力是在 vortek 或者 wrapper 上做一些软件模拟。比如 mali 不支持 bc4 纹理 api，那么就用 cpu 模拟一个，免得出现贴图错误。winlator bionic 和 micewine 的 wrapper 方案都在弄。

窗口集成有三种可能



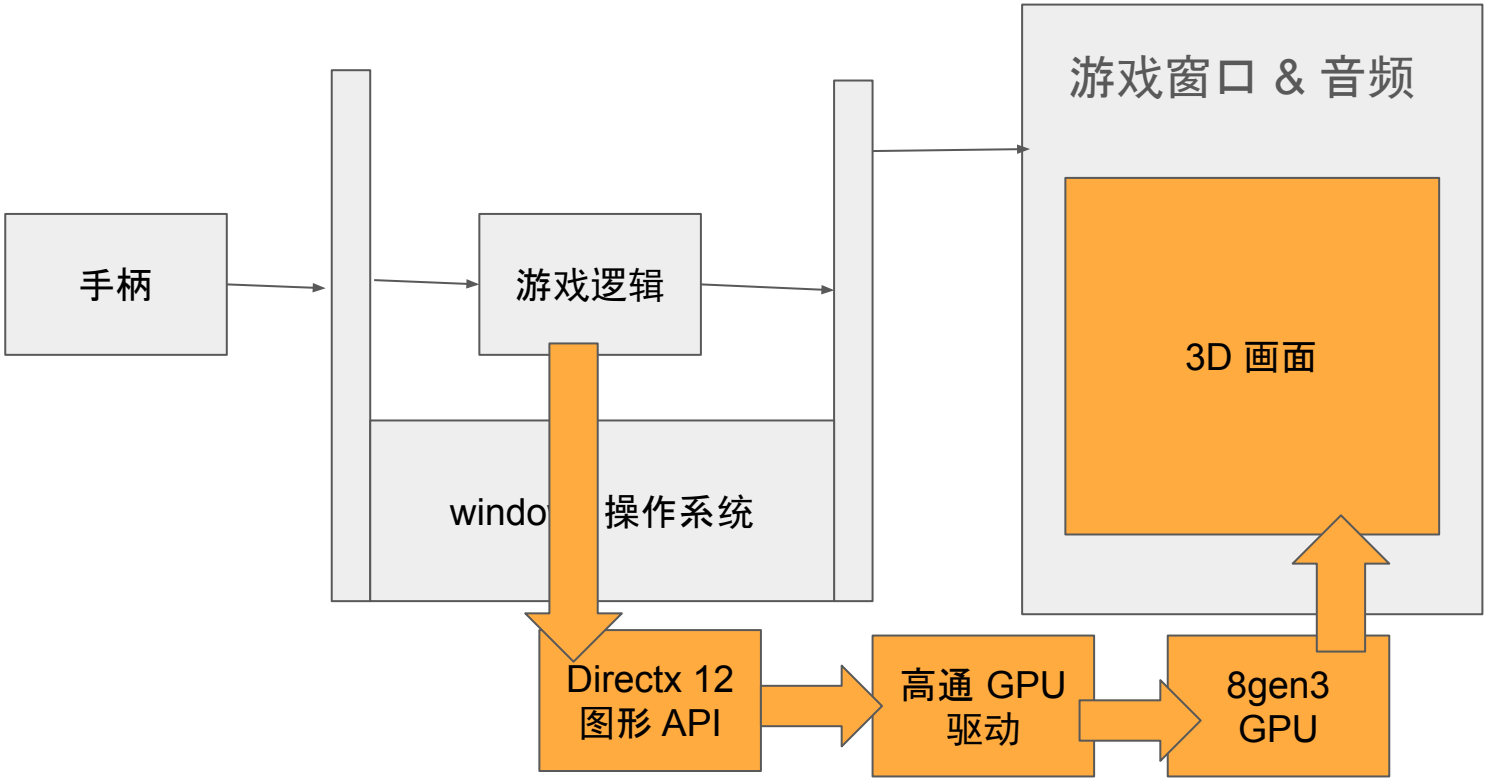
- 直接渲染到同一个窗口上：windows 的窗口和安卓的窗口能一样嘛，完全不是一套东西。整个 windows 多窗口，在安卓下都是一个窗口内的 opengl surfaceview，就是一块大画布
- 先渲染到显存里，然后用 GPU 拷贝到一起
- 先渲染到显存里，然后传输到 CPU，再拷贝到一起：太慢了

vulkan wsi 和 x11 dri3 extension

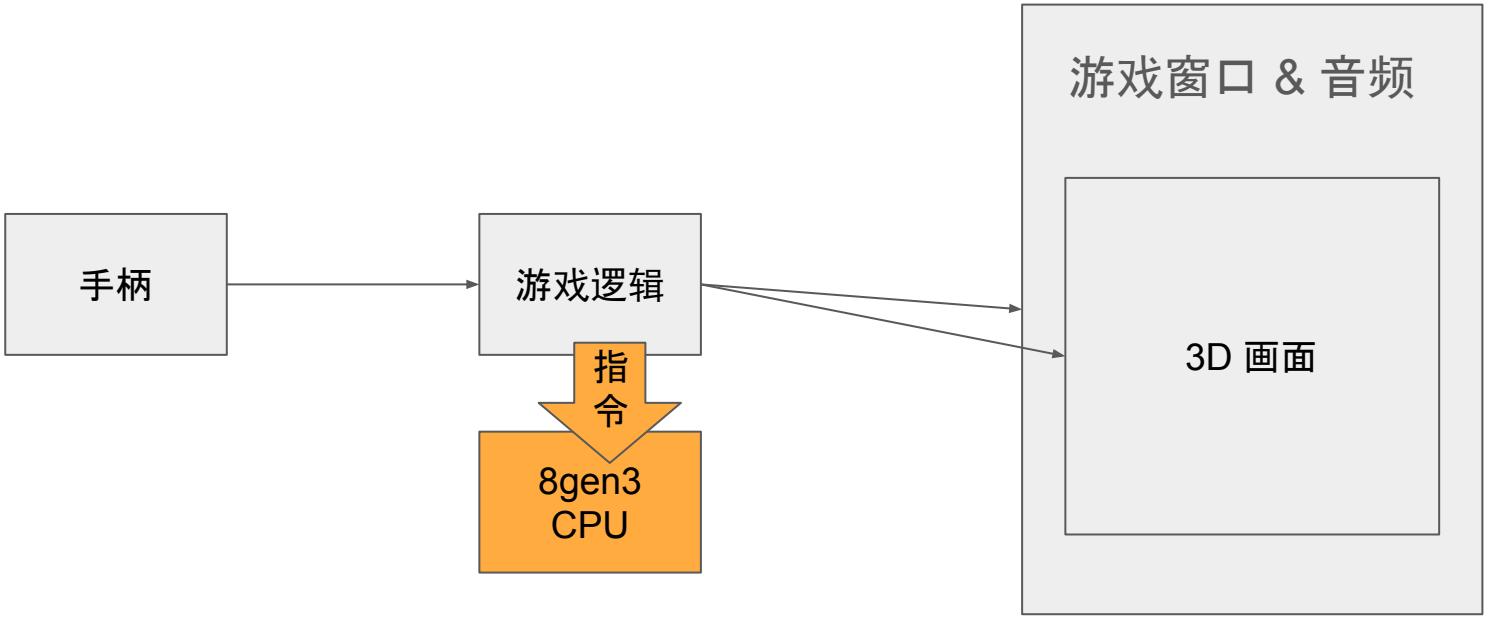


- vulkan 通过 vulkan wsi 从安卓申请一块显存来渲染
- 然后 x11 dri3 允许直接把这个显存上的图片粘贴到 x11 窗口上
- 达到“直接渲染”的效果，不需要 gpu -> cpu 拷贝来拷贝去

图形 API - 更直接控制 GPU：完结

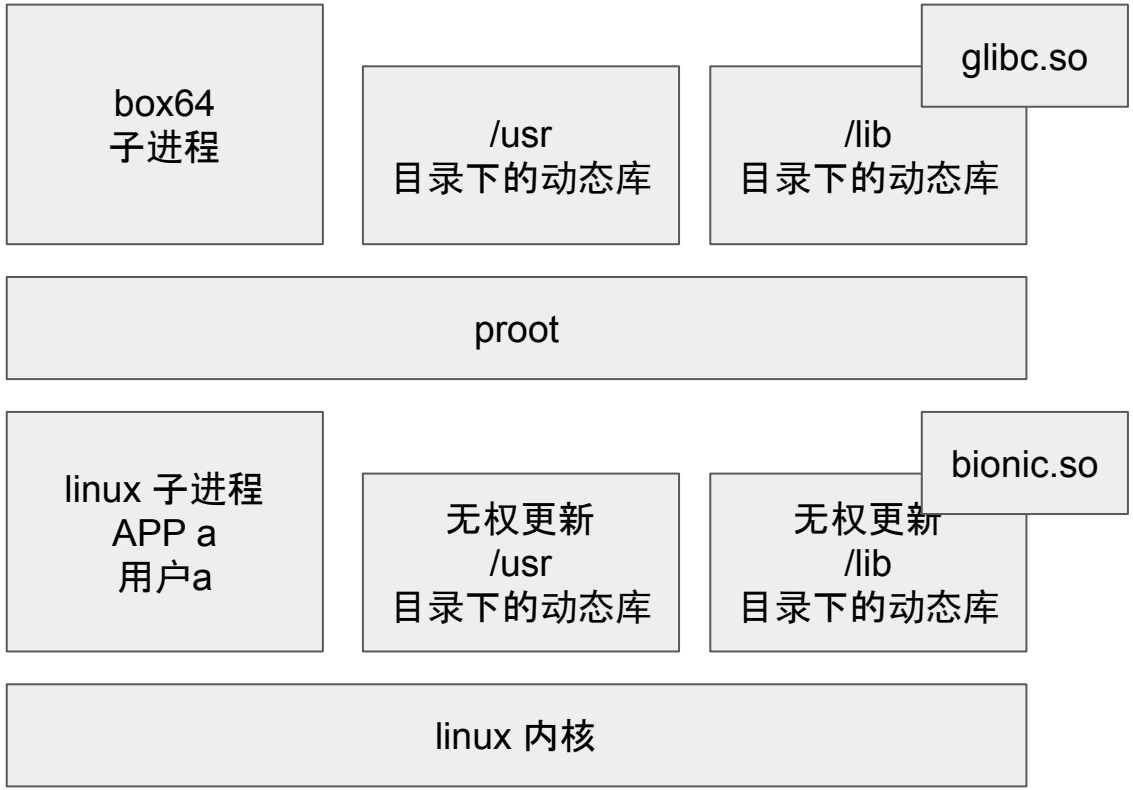


指令集问题的不仅仅有 proot 里跑 box64 这一个方案



手机的 cpu，比如高通骁龙 8gen3 是 arm64 的指令集，但是哈迪斯 2 是 x64 指令集的可执行文件

winlator-proot 非常臃肿



box64 的三种执行方式

- winlator 原版：proot 里跑 glibc 版的 box64
- box64 glibc 版本：去掉了 proot，直接跑 glibc 版本的 box64
- box64 bionic 版本：bionic box64 带了 wrapper vulkan 驱动的支持
  - winlator bionic 版本中的 box64 还是 glibc 版本，并不是 bionic box64
  - micewine 的 box64 是 bionic 的
  - 盖世游戏（gamehub，gamefusion）中的 box64 是 bionic 的

常规 glibc 的 elf 加载过程

ELF（Executable and Linkable Format）是 Linux 下常见的可执行文件格式。其加载过程大致如下：

1、内核加载 ELF 文件

当你在 shell 下执行一个 ELF 可执行文件时，内核会读取 ELF 文件头，判断它的类型（可执行文件、共享库等）。

2、查找动态链接器（Interpreter）

如果 ELF 是动态链接的（即依赖共享库），ELF 文件头的 PT_INTERP 段会指定一个动态链接器（如 /lib64/ld-linux-x86-64.so.2）。内核会加载这个链接器，并把控制权交给它。

3、动态链接器加载依赖库

动态链接器负责解析 ELF 的依赖库（如 libc.so、libm.so 等），并加载到内存中，修正符号地址。

4、执行入口点

所有依赖库加载完成后，动态链接器跳转到 ELF 的入口点，开始执行程序。

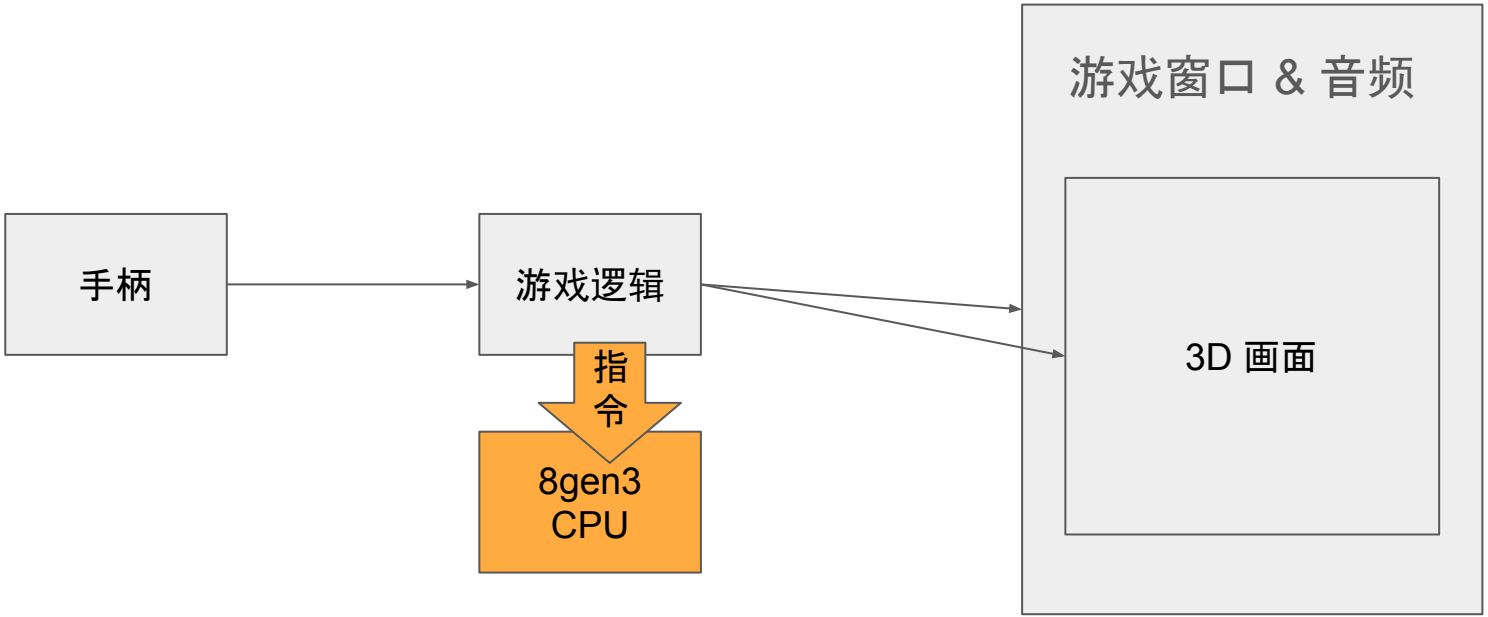
去掉 proot 的原理是什么？

- 修改 elf 文件，指定用 glibc 加载
  - `patchelf --set-interpreter /data/data/com.termux/files/usr/glibc/bin/ld.so your-binary`
- 修改 elf 文件，指定动态链接库从什么目录加载
  - pathelf –set-rpath xxx
- termux glibc packages: [termux-pacman/glibc-packages: Glibc packages for termux]

box64 的三种执行方式

- winlator 原版：proot 里跑 glibc 版的 box64 动态链接 ubuntu 的库
- box64 glibc 版本：动态链接 termux 的 glibc 库
- box64 bionic 版本：动态链接 termux 和安卓的 bionic 库

指令集问题的不仅仅是 x64，还有 x86



手机的 cpu，比如高通骁龙 8gen3 是 arm64 的指令集，但是哈迪斯 2 是 x64 指令集的可执行文件

box86

- box64 是 x64 翻译成 arm64
- box86 是 x86 翻译成 arm32

box64 方案下的 x86 支持

- wow64: box64 里套娃 wow64
- box86: 非纯 64 位的 gpu，可以使用 box86 来跑 x86 应用：关闭 wow64 选项
- box32: box64 的特殊 box32 build 版本 [Release Winlator 7.1.5 with Native Glibc Box32 Mod 2 · alexvorxx/winlator (github.com)]

box64 方案的缺点

- box64 官方没有提供 bionic 的支持，需要第三方补丁
- box64 是整体转译了 wine 里的 exe 和 dll。其中 exe 是游戏，dll 很多是 wine 的翻译层代码，以及图形 api 的翻译成代码。比如 dxvk 和 vkd3d 都是 dll。这些 dll 都会被 box64 从 x64 翻译成 arm64。
- 更好的方案是：直接 bionic 支持，并且仅仅翻译 exe 部分，让 wine 的 dll 直接是 arm64 指令集的

wine 的 arm64ec 模式，仅支持 fex

box64 的 arm64ec 模式还在开发中

wine arm64ec 模式的文档

- wine 的文档：ARM64EC Toolchain · Wiki · wine / wine · GitLab (winehq.org)
- fex 的文档：Development:ARM64EC - FEX-Emu Wiki
- Proton 9.0 ARM64EC – Best Termux-Based Windows Emulator For Android! (zbeon.com) Proton 9.0 ARM64EC – Best Termux-Based Windows Emulator For Android!
- termux proton: tur/tur/proton at master · airidosas252/tur (github.com)

wine amd64ec 作者：CodeWeaver 的 CTO

Alexandre Julliard

Chief Technology Officer
Lausanne, Switzerland

Alexandre Julliard was one of the first developers of Wine when it started in 1993. In 1994, he assumed the responsibility for maintaining the Wine project, and has led the project ever since.

Alexandre graduated in 1994 from the Swiss Federal Institute of Technology in Lausanne, Switzerland with an MS in Computer Science. He then worked for various companies developing embedded software for devices like payphones and routers, all the while maintaining the Wine project in his spare time.

In 1999 he joined CodeWeavers to work full time on Wine, assuming the role of CTO in 2003.

wine amd64ec 也支持 x86

- x86 -> arm32
  - box86
- x64 -> arm64
  - box64 模拟器
  - fex 模拟器
- x86 -> arm64
  - box64 的 box32 模式
- x86 -> x64 -> arm64
  - box64 下套娃 wow64
  - fex 下套娃 HODLL=box64cpu.dll
  - fex 下套娃 HODLL=libwow64fex.dll

- windows下仍然有大量 32 位应用
- 8 gen3已经是纯 64 位 arm cpu 了

各种版本的 windows 模拟器的 cpu 内核差异

- winlator bionic: wine arm64ec + fex
- 盖世游戏
  - wine arm64ec + fex（选择 proton-arm64)
  - box64 bionic (选择其他）
- micewine: box64 bionic
- winlator glibc: box64 glibc
- winlator 原版: proot + box64 glibc
- winlator 10: box64 glibc

选择 windows 模拟器

- 优先选择性能最高的模式，然后降级到能跑的模式
  - 优先选择 winlator bionic 的 wine arm64ec 模式，也就是 bionic 容器
  - 如果跑不起来，选择盖世游戏的 box64 bionic
  - 如果跑不起来，选择 winlator bionic 的 glibc 容器，跑 box64 glibc
- 优先选择 wrapper GPU 驱动，有图形错误降级到 turnip 驱动

box64 / fex 的参数

- 启动不起来，调 safeflags，逐渐改大
- 运行中崩溃，调 strongmem=1，weakbarrier=2
- unity 游戏如果跑不起来，关 callret
- unity 游戏需要调整 fex 的 TSO 模式，从 fastest 改成 fast 或者 slow

感谢 x86 linux windows gaming 社区

- valve steamdeck
- wine + proton
- dxvk
- vkd3d
- vkd3d-proton
- zink
- virgl

感谢 arm mac windows gaming 社区

- mac codeweaver
- wine amd64ec
- AAA gaming on Asahi Linux

感谢安卓 windows gaming 开源社区的贡献者

- box64: ptitSeb ptitSeb/box64: Box64 - Linux Userspace x86_64 Emulator with a twist, targeted at ARM64, RV64 and LoongArch Linux devices (github.com)
- winlator & vortek: brunodev85 brunodev85/winlator: Android application for running Windows applications with Wine and Box86/Box64 (github.com)
- winlator glibc: longjunyu2 Security Overview · longjunyu2/winlator (github.com)
- winlator bionic: Pipetto-crypto Pipetto-crypto/wine-hangover: Hangover and WineCE can be found here (github.com)
- winlator bionic & Mangohud: coffincolors Release MangoHud for Winlator Glibc - Fixed** GPU/CPU stats · coffincolors/winlator (github.com)
- micewine: Pablo Labs KreitinnSoftware/MiceWine-Application: MiceWine is a project that aims to run Windows applications and games on Android smartphones. (github.com)
- wrapper vulkan驱动：xMem xMeM (github.com)
- hangover termux: alexvorxx alexvorxx/hangover-termux: Hangover runs simple Win32 applications on arm64 Linux and Termux (github.com)
- termux-x11: twaik twaik (Twaik Yont) (github.com)
- turnip: igalia + google + others turnip - Danylo’s blog (igalia.com)
- olegos2: olegos2/mobox (github.com)

盖世游戏，蛋蛋模拟器，至今未贡献一行开源代码

shame on you

站在聚光灯下的未必是英雄