

proj156 一种基于Linux系统运行Android 11的解决方案

高校队伍：广东东软学院 Bitcomet队

项目成员：邹明燊、田梓汎、张阳彬

指导教师：刘翠莲、罗泉

项目说明

1. 目标描述

本项目目标在现有Anbox实现Android 7在Linux平台正常运行的基础上将其进一步完善，实现Android11在Linux（Ubuntu20.04）平台上的运行，还可根据用户需求下载并正常运行第三方软件。Anbox（Android in a box）是一个基于容器的方法，可以在普通的 GNU/Linux 系统上启动完整的Android 系统。它将Android应用放进密封的容器中，无需直接访问硬件或数据，所有硬件或数据的访问都是通过与主机上的Anbox守护进程进行的。由于Anbox 直接跑在硬件上，没有软件模拟层，无需虚拟化硬件即可运行 Android，因此可以无缝桥接硬件加速功能。

本项目主要基于Anbox来参考实现，旨在完成以下四个目标：

- **目标1：**完成针对Android11的SDcard的文件系统挂载机制、SELinux安全机制、非特权模式下运行处理对应在Docker容器环境内的处理。
- **目标2：**配置**安卓项目工程**，实现并配置好各个需要的系统模块和AIDL、HIDL、HAL模块，搭建初步调试与测试的Demo1版本，使用adb配合Scrcpy对内部Android远程访问以方便有个初步预期，并且方便本项目有个初步测试的环境。
- **目标3：**完成**Anbox通信部分**实现向Android 11的移植，使Android能与Anbox外部实现跨系统环境通信。
- **目标4：**完成**Anbox其余部分的各个模块移植**，使Android的三个基础部分OpenGL、HWC图形输出、键鼠输入能正常工作。

目前，我们的赛题目标完成度如下：

表1.1 赛题完成度

目标编号	基本完成情况	额外说明
1	基本完成 (≈85%)	1. 与SDcard以及SELinux组件相关的Android 11的基本功能、SDcard文件管理、安装第三方软件、闹铃与联系人等基础应用测试均通过。 2. 非特权模式仅做了部分处理，未进行测试。
2	基本完成 (≈90%)	1. 搭建好相关容器，进行相关基础测试并通过。 2. 目前网络部分由于在Docker特权模式下运行，可能会有Bug。

目标编号	基本完成情况	额外说明
3	初步完成 (≈80%)	1. 完成跨系统环境通信，安卓中Anbox的相关模块可以使用QEMU_PIPE（实际容器中是没有QEMU设备，Anbox实际用了Unix Domain Socket代替，但其通信的相关通道仍然叫做QEMU_PIPE）进行通信或者使用Anbox实现的RPC调用。 2. Anbox的OpenGL ES的emulation库实现初步测试能通过该通信创建QEMU_PIPE连接，实现RPC调用获取到主机侧支持的OpenGL信息。 3. 由于各个部分的模块暂未完全移植好，还需要全部移植才能测试所有功能。
4	初步测试 (≈60%)	1. 已经把相应的Anbox的各个模块的移植到Android中，其中GPS与Audio模块使用了谷歌给Goldfish的实现。 2. OpenGL ES实现和HWC实现在Android 11下无法测试，由于Android 11的相关变动，导致输出画面调用了Gralloc的PostFB去输出，而在Anbox中是未做这部分RPC调用的，输出画面失败。 3. 把相关方案在Android 10上进行测试，测试可以输出画面，但卡在安卓Launcher第一屏画面。
总计	≈80%	还有更多的工作需要测试和完成。

2. 比赛题目分析和相关资料调研

2.1 题目分析

(1). 题目旨在实现Linux的运行Android11的解决方案，在Linux上基于容器技术隔离一个环境运行Android，其运行机制是直接基于宿主机内核下运行Android，其运行效率非常高效，对系统资源占用和运算资源的损耗都极低。

(2). 在所有实现方案中，选定了基于Anbox的实现方式。由于现今用户使用的软硬件环境不统一，Mesa 3D等实现方式对图形处理硬件的支持有限，用户的图形环境不一定是Wayland而是X11偏多。同时考虑到可能未来需要在各类国产主机运行架构不统一（Arm、x86、MIPS、LoongArch等架构），**Anbox对于不同系统不同硬件的兼容性优势使得其是现有方案中最合适的。**

(3). 在实现过程中，考虑到Anbox的LXC也是比较老的容器架构，同时其集成LXC的操作不利于后续容器核心升级、配置调整、在线镜像更新和镜像快速部署等，所以使用Docker来支撑其Android的运行环境，对比最初的LXC方案来说拥有更高的灵活性和可靠性。

(4). 项目计划是初期构建调整好Android高版本核心跑起一个基于Docker容器实现的Demo，后续逐步移植剩余Anbox组件到Android11上。

2.2 资料调研

2.2.1 现有方案

(1). Anbox：基于LXC容器运行Android 7，利用QEMU pipe实现Android与Anbox上层Linux软件用户接口进行通信。其通信中输入的数据包括：接收传入的传感器数据，接收用户鼠标点击和触摸事件，接收APP启动和窗口调整数据、Anbox上层软件屏幕缓冲区的可选回传等；通信中输出的数据则包括：传输Android上2D图形画面和OpenGL的渲染指令到Linux环境中渲染、传输音频输出数据。Anbox运行Android的机制的实现方式就是通过将Android内必要的输入输出交给外部Linux环境处理，**其实现接口均通过软件实现，这样对于兼容性容易出现问题的OpenGL加速部分尤为友好。**但是也正因为如此，虽

然保证了其兼容性，但是其支持的OpenGL库选择就非常有限，目前仅支持OpenGL ESv2，性能和稳定性会较为一般。总结其优缺点就是对于各类操作系统兼容性好，外部有D-Bus和OpenGL环境基本就能正常运行，缺点是通过QEMU pipe传输到外部渲染的方式，其模拟的库只支持OpenGL ESv2，**稳定性一般，性能也比较逊色。**

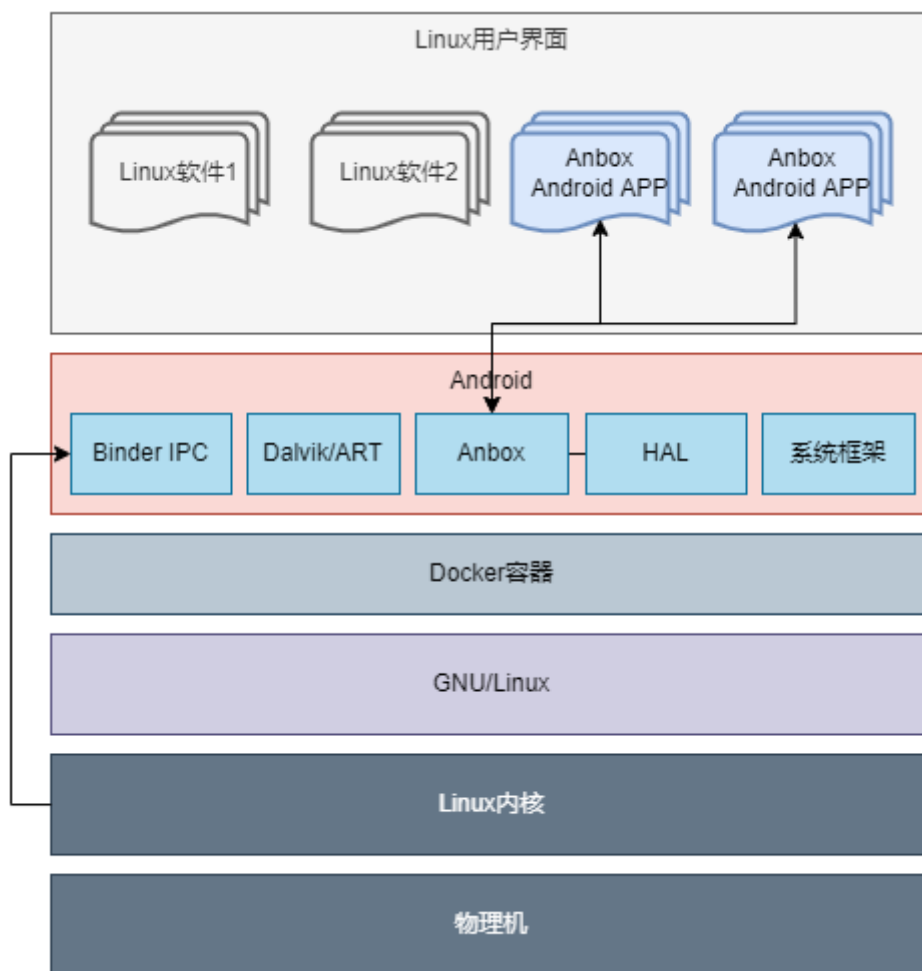


图2.1 Anbox实现架构图

(2). Waydroid: 基于Anbox繁衍而来的实现方案，Anbox除了2018年后面稍微更新了下支持Snap和Anbox Cloud外，实现的方案已经多年未进行更新，核心无较大变化，所支持的最新Android系统仍然是2016年11月24日发布的Android 7.1.1。因此Waydroid应运而生，其前身是基于Anbox的中期版本，基于重建脚本、更新的LXC3、Mesa 3D、最新的Android8-11版本、去掉Anbox代码，后续演化后改名Waydroid。Waydroid更加新颖和完善，其优势在于最新的Android、直接通过Mesa 3D驱动显卡、支持Wayland环境；缺点是**只支持Wayland，Mesa 3D作为开源驱动，只支持部分显卡。**

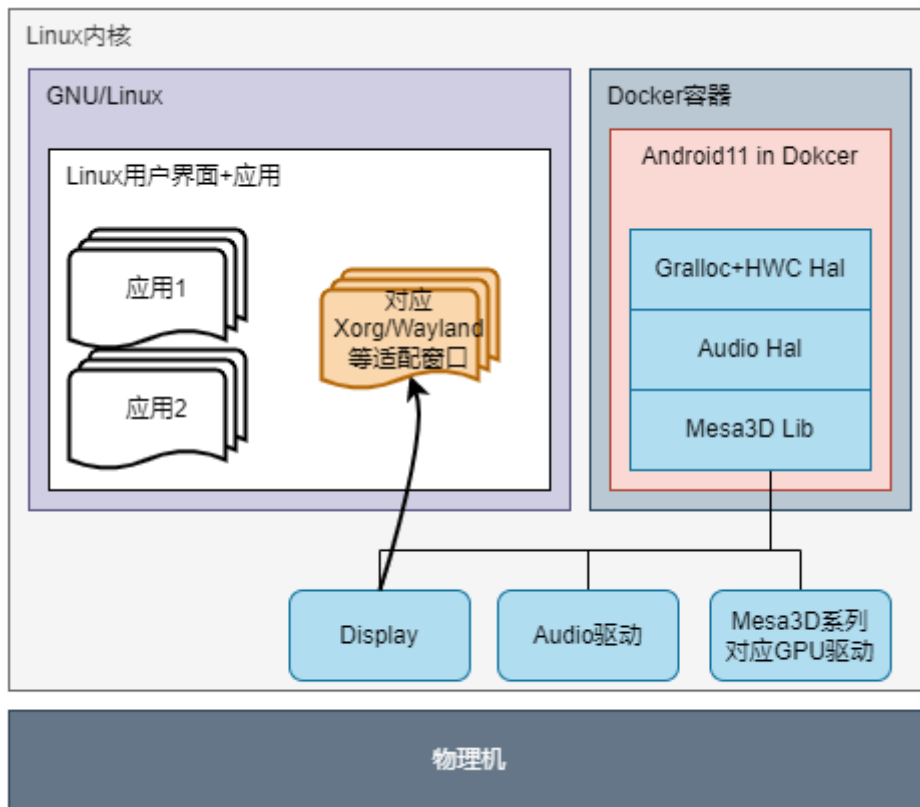


图2 Docker运行Android+Mesa3D架构图

(3). 其他方案：xDroid、Kydroid、KMRE等第三方闭源方案：xDroid和Kydroid也是类似Anbox的方式，其核心原理仍然是通过容器技术让Android直接运行在Linux上，以Linux原生程序运行，由于其闭源，能够了解的公开信息不多。KMRE从其论文（参考借鉴3）来看是类似Waydroid的Mesa3D加速的方案，其性能损耗低，但其一大特色优点在于其对接了现在国产系统常见的Xorg环境下的桌面环境，同时做好支持触摸、优化各类输入输出、传感器等的实现，对比以上开源项目，KMRE不但只实现了基本需求，而且在人性化交互等方面，易用性更高。

表2.1 方案对比

现有方案	优点	缺点
Anbox	对各类环境兼容性好、原生运行速度快	稳定性一般、其所支持的Android版本比较老旧、且Android内只支持OpenGL ESv2,图形化API老旧
waydroid	最新Android、支持Wayland、直接驱动显卡	只支持Wayland图形环境、Mesa 3D开源驱动只支持部分显卡，兼容性一般
其他闭源方案	闭源或商用的项目在用户交互等方面做到了更优，易用性更高	闭源，能查到公开的信息不多

2.2.2 项目需求和状况

(1). 项目现状：

由于西方加速技术封锁的态势，我们必须迫切的找到相应的替代方案来给国内用户有选择的权利。虽然国产Linux系统目前在飞速发展，但其应用生态紧缺，甚至一个输入法、Linux版的QQ这种基础性应用都Bug百出，使用体验极其糟糕，导致用户普及度始终不能有明显的上涨，用户也不情愿使用国产Linux作为替代系统。

因此我们需要一些**中期可替代性方案**，Linux上部署安卓这一个想法应运而生，我们可以利用安卓作为一个暂时的方案来承载现在的业务需求例如办公，视频通话等。安卓在2020年的中国移动操作系统市场份额占比达75.98%，相当于每4个使用智能手机的人中就有3名安卓用户。因其用户整体基数大，所以针对安卓应用开发者非常多，面对安卓衍生出来的各种业务层出不穷，整个生态对于国产Linux有很大的优势。

我们可以直接**借助这个优势**，在Linux上运行安卓系统，完成国产Linux跟安卓生态的整合，进而发展国产Linux，并对其产生助力。而同期也比较少同类型的厂商成功实现在GNU/Linux系统中运行着整个Android系统，市场上有空缺的需求等待填补，我们的项目就是要针对这个市场的空缺，成功实现在Linux系统中运行Android系统而不是单纯的模拟，为我们之后国产Linux和国内安卓的生态整合打下坚实的基础。

(2). 项目需求：

单纯的Linux**无法**成功运行安卓系统中的各种应用程序，因为安卓在Linux层面上主要增加了ART虚拟机和其Application Framework框架，以及其他细枝末节的修改和优化。我们的项目需求就是需要通过一个安卓**中间层**在Linux上提供安卓运行环境，使安卓当中的应用程序也可以运行在我们所配置的容器当中。但是却不是以模拟器的方式，而是使用当前Linux内核基于**容器技术隔离**来运行安卓系统，不需要大量的软件模拟，又保证了安全性。

我们选择了这种方案，对各类硬件资源可以被无缝桥接，让性能的效率损失降到比较低的水准，完全可以满足我们日常需求。该方案补充了当前国产Linux应用生态，可以实现国产Linux操作系统应用生态不足的前提下需要快速扩充用户数量的愿景，让国产Linux操作系统走入平民百姓家。

3. 方案设计

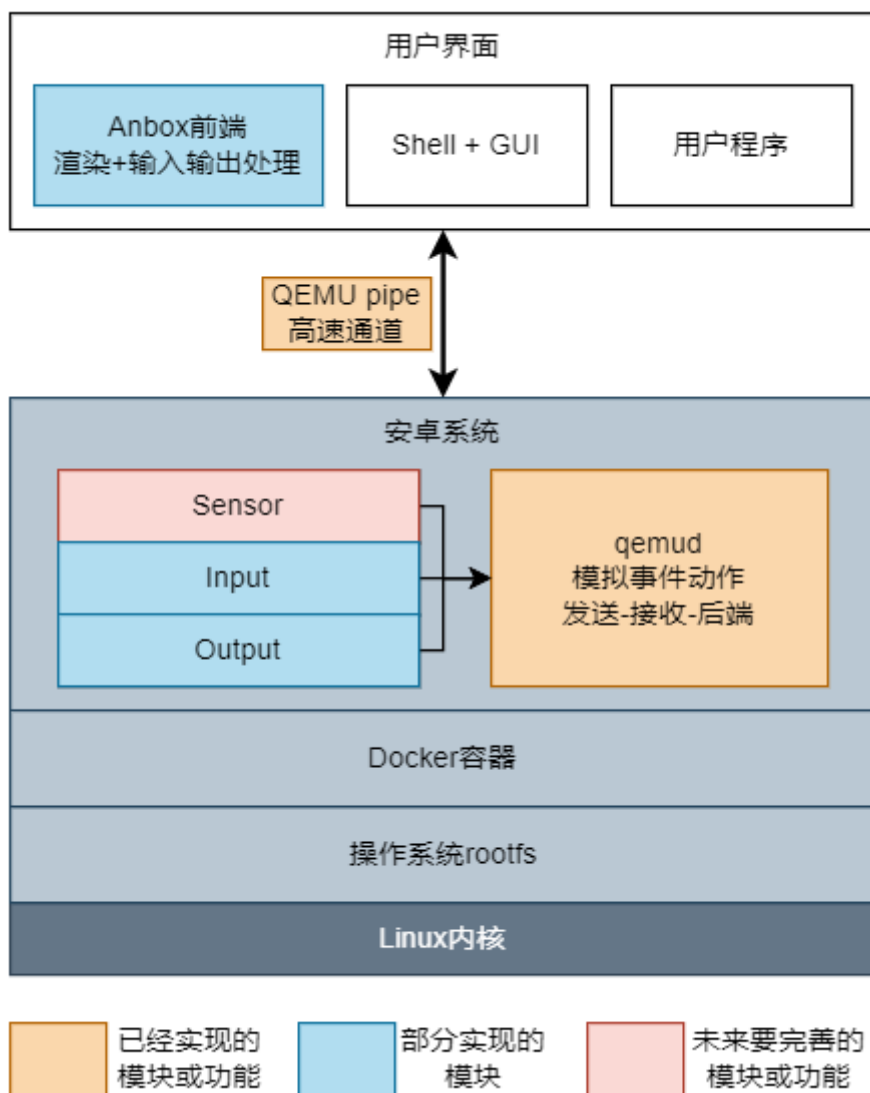


图3.1 Linux系统运行Android 11的解决方案架构图

此项目基于原有Anbox方案进行**更新及改进**。我们**改进**使用基于Docker容器技术提供隔离的、与Linux并行运行的安卓环境，此种改进是对原有LXC容器的**更优替代**。但是该改进需要重新适配与调试，需要大量移植与测试的工作量才能保证维持原有稳定性，并更好地利用更多的新特性开发新的功能。除此之外**更新**新的Android 11替代原有Android 7，此过程需要重新移植原有Anbox的各个组件适配新安卓上的HIDL接口。同时需要修改Android底层，经过大量的移植和测试的工作，让其能正常运行在Docker容器中。此项目利用Anbox部分现成的模块，在Linux下上层APP实现与安卓底层对接输入输出和OpenGL ES的渲染等实现。这些设计目标在于使用Docker**替代**原有Anbox使用的较旧的LXC容器技术，并**更新**其Android 7到Android 11，其中技术革新点如下：

- (1). 使用更新的Android 11替代原有老旧的Android 7。
- (2). Android 11中的HAL层使用了HIDL接口这一新的Treble的架构，更加规范及更高的可移植性。
- (3). 使用生态丰富、快速部署、可移植、更加可控的Docker作为底层容器运行环境替代原有集成LXC这一高耦合、低可移植性的实现方式。
- (4). 活用成熟的Anbox部分对多平台高兼容性的实现方式，以应对目前国产系统多种硬件平台和系统环境的问题。

在以上的设计方案下，我们针对其各个部分进行详细设计，详细方案见“**Bitcomet设计文档**”。

- (1). 底层部分：Docker下的安卓11容器设计：

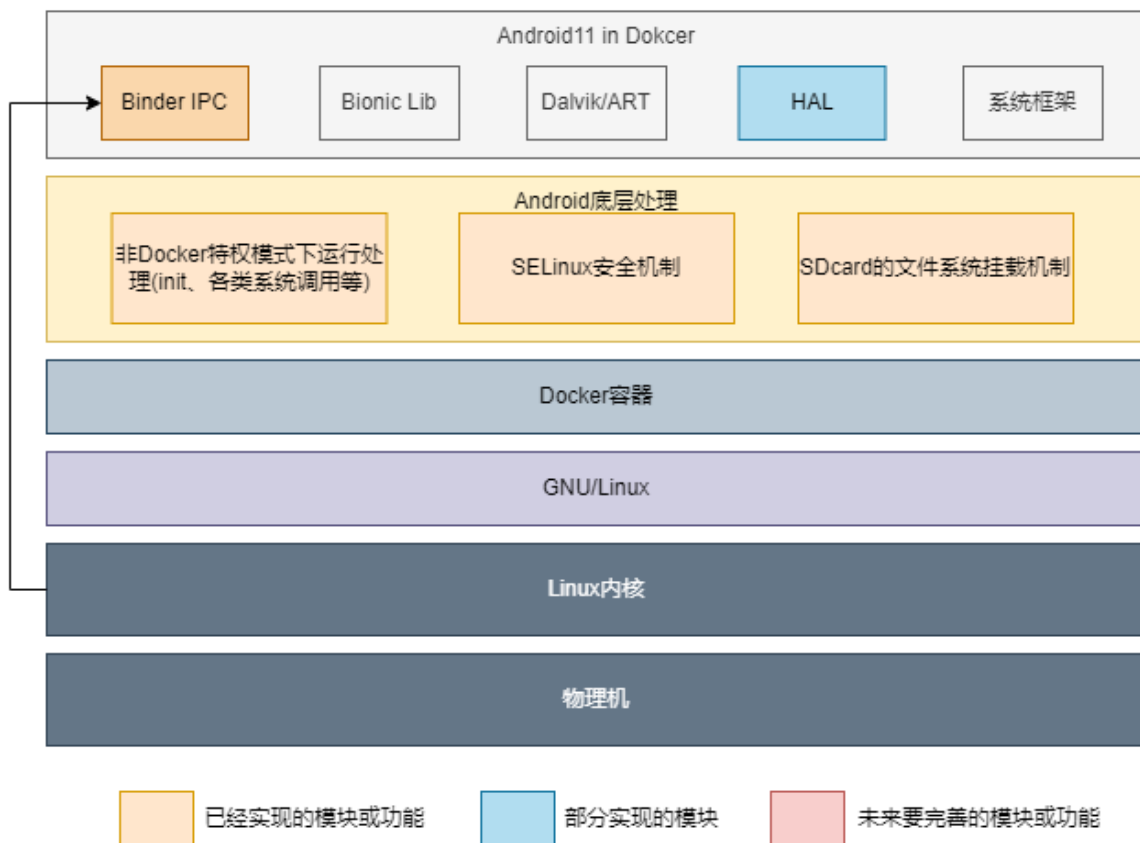


图3.2 Bitcomet实现底层部分设计框架图

• **高可移植设计**：使用了Docker便集成了Docker相关的优点，Docker相比原有LXC有可移植性、版本控制、回滚、快速部署等优点。同时原有LXC容器与Anbox进行了高度集成，在目前多平台和可移植性考量下，原有Anbox的系统镜像与各个配置选项都不能方便地进行更改，这将会导致我们适配到国产系统中遇到重重困难。

• **快速迭代更新**：使用Docker提供底层容器运行环境，利用其快速部署的优点，可以使Android 11更加灵活地运行在目标平台。

• **优雅的数据管理**：利用Docker的版本控制和回滚特性，设计恢复出厂设置并清空用户数据的功能，并为以后Android镜像更新时能够快速部署，及时把新版本推送做准备。

• **较高系统安全性**：把安卓放在密封的容器当中，其外部访问数据都通过QEMU pipe交给上层Anbox前端应用接管，其应用也只有基本的输入输出功能，安全性较高。

• **极低的运行损耗**：通过容器直接基于现有系统Linux内核运行，并基于Linux内核特性进行容器隔离，这一方案提高了稳定性并降低了运行损耗。

(2). 中间层：安卓中运行qemud提供QEMU pipe这一高速管道与上层通信：

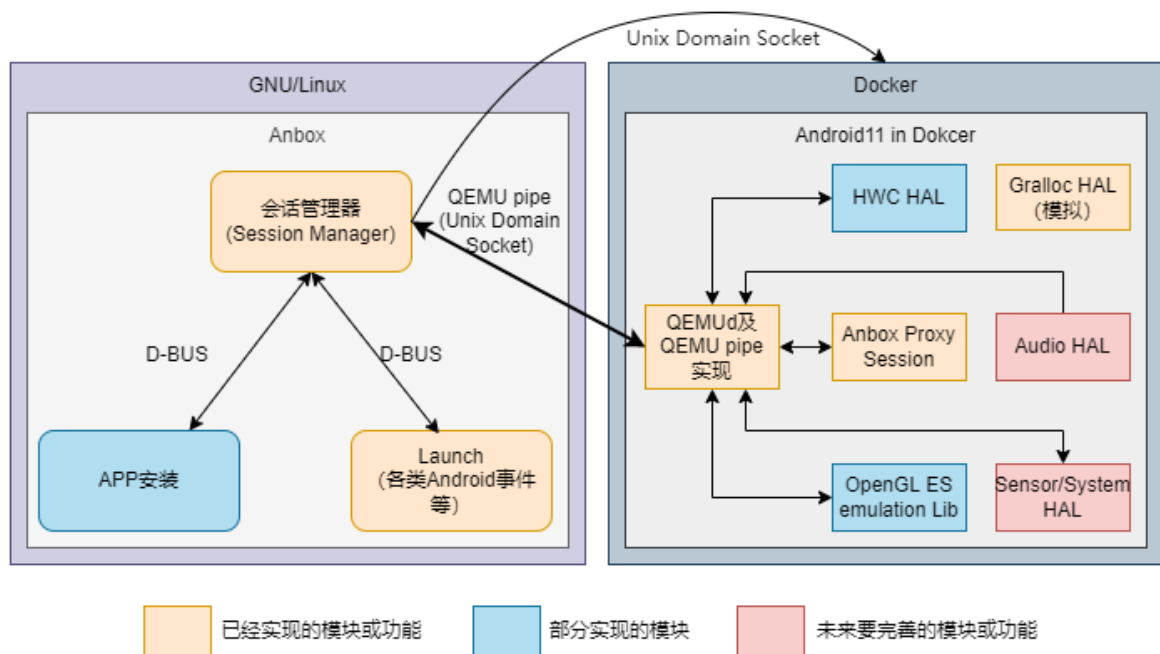


图3.3 Bitcomet中间层部分设计原理图

• **成熟的通信方案：**这一通信方案基于成熟的Anbox的通信这一部分的功能实现，其上层Session Manager提供接口，由安卓和Anbox两端利用相应API把接口打开进行通信。其实现方案最早可以追溯到2013年的谷歌Goldfish的Android模拟器实现并且沿用至今，实现了Android与Linux的之间通信的兼容性与健壮性并存的实现方式。

• **便捷的交互方式：**本方案在安卓与Linux之间的通信方案设计基于QEMU pipe这一高速通道，直接在Android与Linux之间打通一个灵活的通信渠道。本方案不用考虑需要的具体通信实现，直接在安卓HAL层服务和上层Anbox前端应用中打开相应通道进行安卓与Linux对接的数据通信。

(3). 上层部分：Anbox通过QEMU pipe与底层通信，接收安卓的渲染信息渲染以及音频在上层APP输出，把上层前端APP的输入和状态传递给安卓。

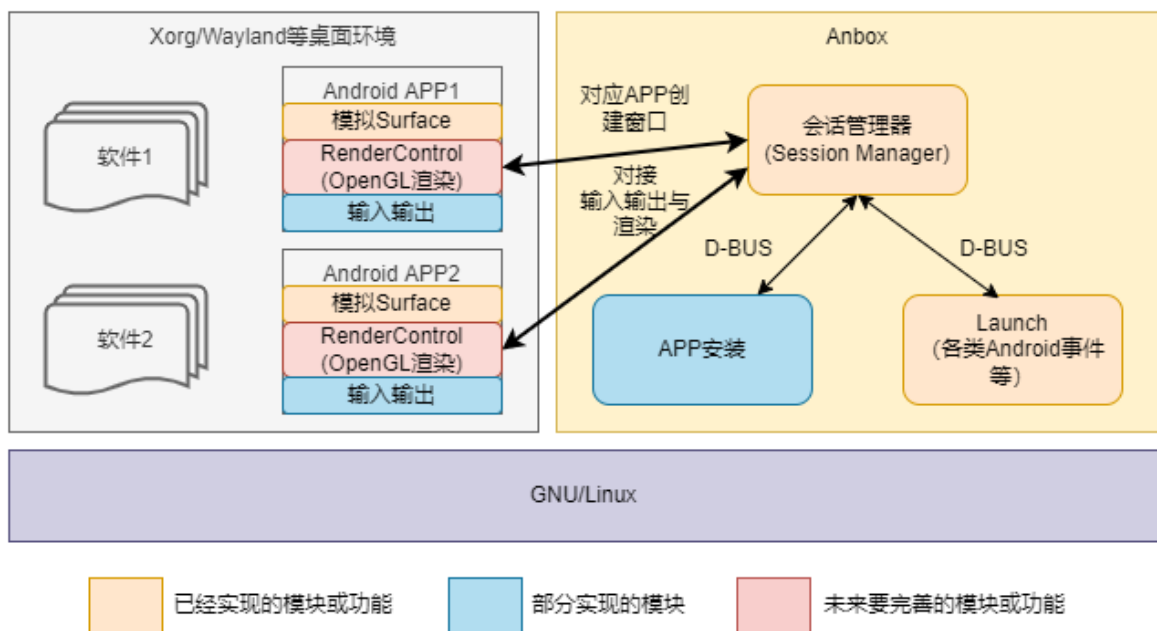


图3.4 Bitcomet上层部分设计原理图

• **兼容性高：**主要体现在Linux下对Anbox的兼容性，这一方案的设计把安卓需要与Linux交互和获取的数据都通过QEMU pipe传输给Linux处理，上层Anbox前端不依赖硬件。尤其是渲染也是调用系统现有OpenGL库实现，兼容性相比虚拟机来说更高。

• **易用性高：**通过上层Anbox应用接管底层安卓的输入输出等部分，相当于把安卓应用直接映射给

了Linux应用。这一操作安卓应用非常接近使用Linux应用的方式，极大的提高了其易用性，操作起来也更加简单。

4. 开发计划

4.1 开发计划表

表4.1 开发计划表

时间节点	内容
5月1日-5月31日	完成安卓11运行在Docker上的核心Demo
6月1日-6月15日	移植Anbox各个组件到安卓11，初步先移植Anbox的OpenGL和输入部分
6月16日-6月30日	完成Anbox的通信组件移植
7月1日-7月31日	完成剩余Anbox组件移植并测试
8月1日-8月15日	完成最终项目提交、整理项目、准备答辩内容

5. 比赛过程中的重要进展

5.1 比赛进展表

表5.1 比赛进展表

时间节点	负责人	完成工作	里程碑
3月7日-3月22日	田梓汎	Android 11内核在docker上运行的实现	
3月23日-4月5日	张阳彬	Android 11组件的改造和规范	
4月6日-5月4日	邹明燊	完成Android工程文件编写和ASOP编译	
5月5日-5月14日	田梓汎、张阳彬	完成Android基础核心功能测试	
5月15日-5月20日	邹明燊	完成Android基础核心功能的运行演示	核心对Docker适配成功完成于5月20日
5月21日-6月5日	全体队员	完成项目文档编写和整理	
6月20日-7月10日	邹明燊、张阳彬	完成Anbox中间层的通信组件移植	成功完成于7月10日并通过基础测试
7月1日-7月15日	张阳彬、田梓汎	完成对Anbox的图形及渲染部分的分析文档	
7月10日-7月25日	邹明燊	完成Anbox中上层各组件移植	

时间节点	负责人	完成工作	里程碑
7月25日-8月2日	邹明燊	完成OpenGL部分渲染的初步RPC调用测试	成功完成于8月2日
8月2月-8月15日	全体成员	完成项目文档编写和整理	

6. 项目测试

测试思路主要分为三步：第一步检测Bitcomet能否成功启动容器内的Android系统，主要检测其Android系统开启是否正常，是否成功进入系统界面。如果第一步成功，则基础系统运行测试正常，可以保证Bitcomet内Android系统的基本运作，遂进入第二步测试。第二步测试主要针对Bitcomet容器内的Android系统主要功能是否正常，这一步的检测将包括三个部分分别针对Android系统的三大主要功能进行测试,其中包括有系统信息测试，基础功能测试以及基础应用测试。通过第二部测试的目的是检测Bitcomet内启动成功后Android系统的整体功能完善性。最后是第三步的测试。上两部分的测试主要是检测系统的基本运行以及功能，这一部分测试主要检测在Bitcomet中运行Android的效率，会通过统一变量同时对比Bitcomet与市面上成熟的Android模拟器的性能差距，从而展现Bitcomet方案不可忽视的优势。

测试环境

本次测试环境硬件参数将统一为下表状态

表6.1 测试环境

部件	参数
系统	Ubuntu20.04
CPU	英特尔 i5-8300H@2.3Ghz
内存	DDR4 16GB (2400MHz)
硬盘	主硬盘 128G SSD 从硬盘 2TB HDD
显卡	英伟达 GTX1050&英特尔UHD Graphics 630

6.1 基础系统运行测试简介以及结果

这一步我们主要通过安装环境→加载内核和挂载文件系统→启动Bitcomet并连接容器内的Android来查看其运行情况。

在根据测试步骤，下载测试镜像，安装环境并加载启动Bitcomet所需模块后，Bitcomet可成功启动，通过相关命令可以进入Android shell模式，QtScrcpy投屏软件可以连接Bitcomet并显示Android界面。系统成功启动，运行正常。

6.2 系统功能测试简介以及结果

这一步测试我们分为三个部分进行，以下为各个部分的测试简介：

- (1). 系统信息测试，主要测试系统设备名。
- (2). 基础功能测试，主要测试各基础主要功能例如默认语言，WIFI等。
- (3). 基础应用测试，主要测试系统默认应用运行情况以及第三方应用运行情况。

测试结果主要如下：

表6.2 系统功能测试结果

成功	失败
系统信息和基础功能	输入(远程)
通讯录、闹铃、浏览器等基础应用以及第三方应用	音频
图形渲染(GPU软件渲染)	蓝牙WIFI

6.3 性能对比测试简介以及结果

性能对比测试环节我们采用Genymotion模拟器来作为对比对象，同时会进行CPU性能对比测试以及内存开销对比测试。CPU性能测试主要使用Android平台主流的基础测试应用Geekbench以及安兔兔AI。Geekbench主要针对于CPU的浮点运算和整数运算部分给出性能量化指标，而安兔兔AI则主要针对CPU中的AI运算部分给出性能量化指标。内存开销测试我们选择各方案仅运行一个Android系统，查看整体内存占用，测出内存开销。

在CPU测试部分，Geekbench测试中，bitcomet成绩为单核4437分，多核14398分，Genymotion模拟器对照组单核4251分，多核12626分。相比于Genymotion模拟器，bitcomet单核领先约4.3%，多核领先约14%。性能提升相当于当今移动端旗舰级芯片高通骁龙865和高通骁龙888的性能差距(性能对比数据来自www.socpk.com)，也就是芯片厂商用一年时间更迭优化出的性能。而在安兔兔AI测试中，Genymotion模拟器得分为48231，bitcomet得分为58840，bitcomet更是取得了约22%的显著优势。

内存测试部分，分别测试仅打开Genymotion启动Android系统和仅打开Bitcomet启动Android系统，不运行任何其他应用程序，记录其内存的开销情况，Genymotion模拟器占用整个系统约3.8GiB的内存空间，而Bitcomet占用仅为2.8GiB。占用内存大小仅为Genymotion模拟器的3/4。

6.4 总结

在系统运行测试，系统功能测试部分，Bitcomet已成功在Ubuntu20.04下运行。除部分功能未完善外，通讯录，闹铃，浏览器等基础应用及系统基础信息均可正常使用与显示，也可根据需求安装第三方软件。

通过Bitcomet与Genymotion在CPU性能测试与内存开销测试的情况对比可知，Bitcomet无论是在CPU利用效率还是内存开销上都优于Genymotion。在虚拟化技术成熟的今天，14%性能提升意味着每一百台计算机平台可以少买12台计算机。每百万可以省下12万，又或是在能耗方面做出改进，我们可以限制机器的运行功耗，降至模拟器同样的性能，但是省下更多的电。与此同时，Bitcomet还只是一个“半成品”，其潜力之大可想而知。

演示视频链接: https://pan.baidu.com/s/1FLokWbiU3WNq_i5bhb7sA?pwd=ew23 提取码: ew23

7. 遇到的主要问题和解决方法

因构建Android需要一定的磁盘空间以及CPU资源，本次比赛项目构建编译主要在服务器上进行。

(1). 问题：在运行Android过程中发现服务器网络环境崩溃，无法连接，在联系机房管理员重启网络后，确认本次网络崩溃由本项目引起。在对本项目可能导致网络崩溃部分进行逐个排查时，发现编译后的Android网络组件在运行时会导致此情况发生，确定该故障为网络组件出现错误所引发。

解决方法：经过网上搜索资料以及多次尝试后，发现取消Docker的特权模式，调整Docker为非特权模式下运行Android11容器则不会出现此问题，或将所编译后的镜像打包下载到本地系统，采用虚拟化方式运行也可正常运行。

(2). 问题：QtScrcpy投屏软件未能搜索到Android设备，排除Andorid系统自身问题后，确认问题在宿主机系统环境上。

解决方法：经过查找资料，发现本虚拟机的网络组件配置有问题，如果宿主机需要用到端口进行操作，需要将虚拟化软件设置开启端口映射，映射adb默认端口5555才能解决，后面重置虚拟机网络组件彻底解决了该问题。

(3). 问题：在Android 11中，移植好Anbox的图形实现部分的各个HAL层，实际测试发现无法启动SystemUI Service。

排查过程：经过Anbox通信层连通性测试，Anbox的通信层RPC调用测试，发现均连通并实现RPC调用获得返回值。排除通信层的问题，遂排查相关日志和流程，发现是Android 11中没有调用Anbox的HWC实现，而走了Gralloc的发送到Framebuffer的实现。计划从模块和安卓工程进行调试并排查问题进行解决。

8. 分工和协作

表8.1 分工与协作

项目	负责
Docker运行Android 11基础核心的实现	田梓汎和张阳彬
Androidlunch工程和项目管理	邹明燊
Android基础核心功能测试	田梓汎
Andox基础组件移植	邹明燊
图形渲染输入输出测试	张阳彬

9. 提交仓库目录和文件描述

表8.2 仓库目录与文件描述

仓库	说明
Proj156-bitcomet	安卓manifests存放位置，项目说明
Proj156_device_bitcomet	安卓lunch工程，源码 device/bitcomet目录
Proj156_frameworks_native	修改的android/frameworks/native 目录
Proj156_frameworks_base	修改的android/frameworks/base目 录
Proj156_system_bpf	修改的android/system/bpf目录
Proj156_external_selinux	修改的android/external/selinux目 录
Proj156_system_netd	修改的android/system/netd目录
Proj156_system_libhwbbinder	修改的android/system/libhwbbinder 目录

仓库	说明
Proj156_system_core	修改的android/system/core目录
Proj156_vendor_bitcomet	增加的android/vendor/bitcomet目录放一些工程额外加入的包
Proj156_hardware_interfaces	修改的android/hardware/interfaces目录
Proj156_hardware_libhardware	修改的android/hardware/libhardware目录
注：由于安卓项目太多，所有项目需要通过repo工具组织管理起来，不需要单独拉取覆盖。	

10. 比赛收获

这次是我们小组第一次参加全国大学生计算机系统能力大赛操作系统设计赛，可谓收获满满。以前在课堂上学到的主要都是一些操作系统的经典理论，如进程管理中的同步互斥、PV操作、进程调度算法、银行家算法、死锁等；在存储管理中的内存分配机制和虚存管理中的各种页面置换算法等。通过这次参赛，不断参加主办方组织的技术培训会，我们学到了很多在课堂上接触不到的知识，开阔了视野，增加了对操作系统最新行业发展方向的动态了解。在比赛的过程中，我们也查阅了大量跟操作系统相关的资料，攻克一道又一道难关，对个人能力也是一个很好的提升，具体收获如下：

(1). 接触到先进的项目管理方式：我们的实践能力得到了很大的提升和锻炼，接触了很多开发环境才会用到的东西比如说利用git实现项目的同步，更加科学有效的管理自己的项目，比之前我们出错了还要一步步试出，打快照来备份是一个非常大的改善，修改问题只需要返回之前的提交节点即可，这些都是之前单纯在课堂上学习接触不来的。

(2). 团队协作能力的提升：这次比赛也同时使我也更加的能够融入整个小组。现在的我对于团队协作有了很大的提升和理解，在团队当中每个人会有不同的分工，要把自己的那一部分给处理好才是重中之重，不能因为某一环节而掉了链子。这跟之前的作业有很大的区别。

(3). 目标导向的做事方式：这次大赛的目标非常清晰明确，也给予了相当的参考，方便了我们规划项目，我们规划能力也得到了锻炼。我们学会了朝着自己的目标一步步设置里程碑，去一步步靠近真正的目标，而不是像之前无头苍蝇一样做到哪里就算到哪里，目标明确之后带来的效率的提升也是实打实的。

(4). 实践能力的提升：在理论知识用于实践的能力上也得到了很大的提高，我们理论知识可以用于对我们搭建出来的初步项目的排错，这些成功的排错又可以反过来复习巩固我们的理论知识，是双赢的操作。

(5). 编码能力的提升：在这次项目当中我们审计了非常多的代码，比在课堂上的磨炼要多得多，对于Android各个模块的理解也有了长足的提升，这些能力的提升只靠单纯的课堂是远远不够的，还是需要这种实践机会。

(6). 开阔视野：大赛带给我的远不止于此，它更是开阔了我的眼界，跟这么多所高校志同道合的人同台竞技更是带来了很强的新鲜感和成就感。

所以非常感谢这次的大赛，本次的参赛经验将作为我们人生中的一次宝贵经历让我们铭记于心，也希望这次大赛我们小组能够取得奖项，为团队，为学校，为未来能在操作系统有一席之地打下坚实的基础。

11. 与导师的沟通情况，与导师等是否建立了微信群或交流通道

加入导师交流微信群，定期听取专题会议并参与讨论。

使用方法

编译Android镜像

1. 下载安装repo

```
# 下载repo，参考了清华源的https://mirrors.tuna.tsinghua.edu.cn/help/AOSP/的帮助
mkdir ~/bin
curl https://mirrors.tuna.tsinghua.edu.cn/git/git-repo > ~/bin/repo
chmod a+x ~/bin/repo

# 临时配置repo使用国内清华源镜像，也可以加入~/.bashrc永久配置
export REPO_URL='https://mirrors.tuna.tsinghua.edu.cn/git/git-repo'
# 在android目录下初始化repo使用当前manifests仓库来下载代码
mkdir android && cd android
repo init -u https://gitlab.eduxiji.net/willzen/project788067-124647.git
# repo开始同步拉取
repo sync
```

2. 编译安卓镜像

```
# 切换进android目录，引入环境变量
cd android
source build/envsetup.sh
# 选择编译工程
lunch anbox_x86_64-userdebug
# 开始编译(-j32表示最大用32线程去编译，请根据现有自己CPU核心数选择)
make -j32
# 编译成功后打包镜像
pack
```

3. 测试运行

注意以下步骤需要root权限

```
# 挂载打包后工程当前目录下出现的android.img镜像
mkdir tmp
mount android.img tmp
# 导入镜像到Docker
cd tmp
tar --xattrs -c . | sudo docker import - bitcomet:latest
# 检查Ubuntu 20.04内核的BinderFS能否正常挂载
mkdir /dev/binderfs
mount -t binder binder /dev/binderfs
# 运行Docker的Android容器
docker run -itd --rm --memory-swappiness=0 --privileged -p 5555:5555 --name
bitcomet bitcomet:latest /init androidboot.hardware=bitcomet
```

参考借鉴

1. Anbox(所有项目仓库): <https://github.com/anbox>
2. 安卓AOSP源码获取来自清华大学开源软件镜像站点
3. KMRE: An Efficient and Compatible Runtime to Execute Android Application on Linux System,
[Date of Conference: 10-13 December 2021]. 10.1109/ICCC54389.2021.9674681