计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(11):21−28 [doi: 10.15888/j.cnki.csa.009333]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 QEMU RISC-V 架构的 OpenHarmony 标准系统移植①

邰　阳[1], 韩昌刚[2], 全　雨[1], 于佳耕[1], 武延军[1]

[1](中国科学院 软件研究所 智能软件研究中心, 北京 100190)
[2](中科南京软件技术研究院 智能软件研究中心, 南京 211100)
通信作者: 于佳耕, E-mail: jiageng08@iscas.ac.cn

摘　要: RISC-V 指令集架构的诞生促进了 RISC-V 硬件平台的快速发展, 因此, 在 RISC-V 架构硬件平台上运行高效且易于使用的操作系统需求日益增加. 面向分布式、开源开放的智能终端操作系统 OpenHarmony 技术正在不断演进, 其生态持续繁荣. 然而, 将 OpenHarmony 适配到 RISC-V 指令集架构平台上面临新的挑战, 包括软件栈和芯片移植的问题. RISC-V 指令集架构平台的软件栈和芯片移植存在新的挑战. 本文提出了基于 RISC-V QEMU 平台的 OpenHarmony 标准系统移植的思路和方法, 通过对关键软件栈进行适配和在 QEMU RISC-V 虚拟化硬件平台上移植图形显示驱动, 成功实现了 OpenHarmony 标准系统在 QEMU RISC-V 虚拟化硬件平台上的正常启动, 并进入系统桌面. 该成果为开发者提供了在 RISC-V 平台上测试应用 OpenHarmony 标准系统的平台, 并为新的 RISC-V 硬件平台上移植 OpenHarmony 标准系统提供了参考.

关键词: RISC-V; OpenHarmony; 操作系统; QEMU; 虚拟化

引用格式: 邰阳,韩昌刚,全雨,于佳耕,武延军.基于 QEMU RISC-V 架构的 OpenHarmony 标准系统移植.计算机系统应用,2023,32(11):21–28.
http://www.c-s-a.org.cn/1003-3254/9333.html

Porting of OpenHarmony Standard System Based on QEMU RISC-V Architecture

TAI Yang[1], HAN Chang-Gang[2], QUAN Yu[1], YU Jia-Geng[1], WU Yan-Jun[1]

[1](Intelligent Software Research Center, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](Intelligent Software Research Center, Nanjing Institute of Software Technology, Nanjing 211100, China)

Abstract: RISC-V instruction set architecture (ISA) has promoted the rapid development of the RISC-V hardware platforms, leading to growing demands for efficient and easy-to-use operating systems running on RISC-V architecture. As a distributed open-source mobile operating system, OpenHarmony continues to evolve with constant prosperous ecology. However, adapting OpenHarmony to RISC-V ISA platforms poses new challenges, including software stack and chip porting. This study presents an approach and methodology for porting the OpenHarmony standard system to the RISC-V QEMU platform. Based on adapting critical software stack components and porting the graphics display driver on the QEMU RISC-V virtualization hardware platform, the OpenHarmony standard system successfully starts on the QEMU RISC-V virtualization hardware platform and enters the system desktop. This achievement provides developers with a platform to test and apply the OpenHarmony standard system on RISC-V platforms and serves as a reference for porting the OpenHarmony standard system to new RISC-V hardware platforms.

Key words: RISC-V; OpenHarmony; operating system; QEMU; virtualization

① 本文由“RISC-V 技术及生态”专题特约编辑邢明杰高级工程师、宋威副研究员、张科正高级工程师以及易秋萍副教授推荐.
收稿时间: 2023-05-26; 修改时间: 2023-06-27, 2023-07-17, 2023-07-21; 采用时间: 2023-07-27; csa 在线出版时间: 2023-09-21
CNKI 网络首发时间: 2023-09-23

现代计算系统依赖于高性能处理器, 而 ISA 作为操作系统和芯片之间的接口规范, 对于设计一个优秀的处理器至关重要. RISC-V 指令集架构由加州大学伯克利分校的 Waterman 等人于 2010 年发起[1], 近年来引起了极大关注, 并逐渐成为与 x86 和 ARM 并列的主流指令集. 作为一个开放、可扩展和安全的 ISA, RISC-V 未来在服务器、个人电脑、智能手机、物联网和其他计算设备中具有相当大的潜力[2].

随着 RISC-V ISA 的快速发展, 面向 RISC-V 的操作系统成为近年来研究领域的热点. 目前, 国内外主流的操作系统发行版本已经陆续支持 RISC-V 架构[3,4], 例如 openEuler、openKylin[5]、Ubuntu、Debian、FreeRTOS、FreeBSD 等, 主要面向嵌入式、桌面和服务器设备. 然而, 由于 RISC-V 是一种新兴的硬件架构, 其操作系统领域在性能、功能和应用生态等方面仍存在诸多问题需要解决. 操作系统不仅要具备有效地管理计算机系统的硬件资源的能力, 还要提供必需的标准功能服务支撑上层应用, 并且需要为用户提供友好的人机交互功能, 以提高系统的易用性. 智能终端设备的数量持续高速增长, 催生出了更加碎片化、多样化的需求和场景, 从而对新一代操作系统提出了更高的要求[6].

OpenHarmony (OH) 是面向分布式全场景、开源开放的智能终端操作系统. 官方标准系统支持 ARM 架构相关开发板平台, 但缺乏对 RISC-V 架构的支持. 自 2022 年以来, RISC-V 硬件生态陆续推出高性能开发板, 如 VisionFive2 和 Lichee Pi 4A, 并且面向操作系统领域诞生了一些 RISC-V 特有的软件栈, 如蓬莱 TEE. 同时, OH 硬件平台的使用需求量不断增大, 目前, 已经有 14 所高校成立了 OH 技术俱乐部. 因此, 为了丰富 RISC-V 软硬生态产品的应用领域并为学生提供一个快速开发、调试和测试的 OH 硬件平台, 构建一个基于 RISC-V 虚拟化平台的 OH 标准系统是非常吸引人的. 然而, 随着 RISC-V 硬件平台的发展, 必须解决许多挑战, 如系统构建、硬件适配和应用程序生态系统.

本文介绍了基于 QEMU RISC-V 架构的 OH 移植技术. 我们首先明确了 RISC-V 平台的 OH 标准系统支持的整体方法, 然后在 RISC-V 架构上移植了 OH 标准系统包括的主要软件栈, 如工具链、编译系统、基础 C 库、ARK 运行时等, 并基于 QEMU RISC-V Virt 平台进行图形显示模块的移植, 从而使 RISC-V 架构平台上的 OH 标准系统成功进入系统桌面.

1 基于 RISC-V 平台的 OH 标准系统

1.1 OH 标准系统介绍

OH 是一个旨在构建一个全场景、全连接的智能终端操作系统. OH 的设计目标是支持多种设备类型, 包括手机、平板电脑、智能穿戴设备、智能家居设备、车载系统等. 它提供了一个统一的开发平台和开发框架, 使开发者能够更轻松地开发应用程序, 并且能够在不同设备上无缝运行[7].

OH 支持轻量 (面向 MCU 类处理器例如 ARM Cortex-M、RISC-V 32 位的设备)、小型 (面向应用处理器例如 ARM Cortex-A 的设备, 支持的设备最小内存为 1 MiB) 和标准系统 (面向应用处理器例如 ARM Cortex-A 的设备, 支持的设备最小内存为 128 MiB), 这种划分允许 OH 适配不同级别的硬件设备, 使其具有广泛的适用性和灵活性. 本文工作只面向标准系统, 其软件栈主要包括内核、硬件抽象 HDF、系统服务层、框架层和应用层.

1.2 RISC-V 平台的 OH 标准系统架构

目前, 面向 Ubuntu、Fedora 和 openEuler 等 Linux 操作系统的 RISC-V 架构移植主要面临两大难点, 即内核适配和广泛的软件包支持. 内核适配需要确保硬件平台通过引导加载程序正确运行内核, 并支持 RISC-V 架构的特性和功能. 由于这些 Linux 操作系统拥有广泛的软件包生态系统, 为适配新的架构, 需要重新编译和调整软件包, 并投入大量的人力. 虽然 OH 标准系统同样基于 Linux 内核, 但作为智能终端操作系统, 依赖于大量的设备驱动程序, 包括图形、摄像头、无线网络等. 在 RISC-V 架构上移植 OH 时, 需要重新编写或调整这些驱动程序, 以适配新的架构和硬件设备. 此外, OH 标准系统为不同硬件平台提供了特定的功能和优化, 如传感器、GPS、移动网络等, 移植时需要解决这些平台特定功能的支持. 最后, OH 拥有其特定的应用程序生态系统, 无法参考其他 Linux 操作系统应用移植. 综上所述, 与常见的 Linux 操作系统相比, 大量的驱动适配和不同硬件平台的定制化适配导致 OH 标准系统的 RISC-V 移植难度更高.

另外, OH 标准系统在 RISC-V 架构平台还面临许多挑战. 首先, RISC-V 软件生态目前还处于发展阶段, 与 x86 和 ARM 相比, 指令集相关的软件栈功能尚不完

善. 因此, 我们需要对 OH 依赖的软件栈进行功能的移植, 尽可能与 ARM 架构对齐, 以确保 OH 能够在 RISC-V 平台上正常运行. 其次, OH 标准系统提供了增强的交互能力、3D GPU 以及硬件合成能力, 然而, 目前满足这些需求的 RISC-V 开发板较少, 并且相应的驱动模块不完善.

为了解决上述提到的挑战, 我们提出了 OH 标准系统 RISC-V 移植架构, 见图 1 所示. 首先, 面对 RISC-V 硬件平台数量有限且性能弱的问题, 我们选择基于 QEMU RISC-V 虚拟化平台进行移植, 并采用 VirtIO 虚拟化驱动模块作为设备驱动, 以实现与 QEMU RISC-V 架构的驱动对接. 其次, OH 标准系统在 RISC-V 架构上的支持主要集中于架构强相关的软件栈, 比如编译工具链、编译框架、musl libc 库、ARK 运行时, 以及其他依赖包, 包括 unwind 解析库、编解码库和图形库等. 最终, 在 QEMU RISC-V 虚拟化平台上成功启动了 OH 标准系统, 并运行了系统基础应用, 例如设置、时钟和计算器等.

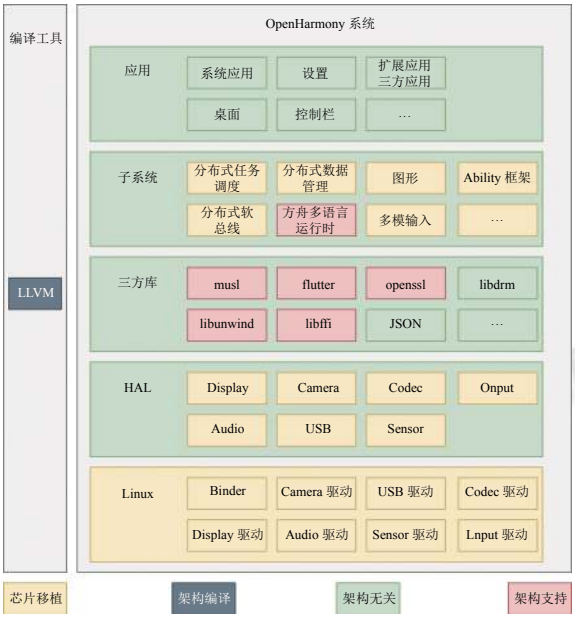

图 1 OH 标准系统 RISC-V 移植架构

2 QEMU RISC-V 架构的 OH 标准系统移植

2.1 架构强相关软件栈移植

为实现 OH 标准系统软件栈在 RISC-V 架构上的移植, 首要工作是确保工具链对 RISC-V 的支持和适配编译子系统, 以满足 OH 所有软件栈在 RISC-V 架构下的编译构建需求. 其次, 我们通过关键字索引的方法定位 OH 标准系统中与架构强相关的软件栈, 明确当前 OH 版本需要移植的代码仓列表. 这些架构强相关的代码主要涉及编译脚本、汇编代码以及与架构相关的函数实现和宏定义等. 在完成所有架构相关的代码仓适配后, 我们即可编译构建适用于 RISC-V 64 架构的通用软件包, 为后续的芯片移植工作提供支撑. 整体流程详见图 2.

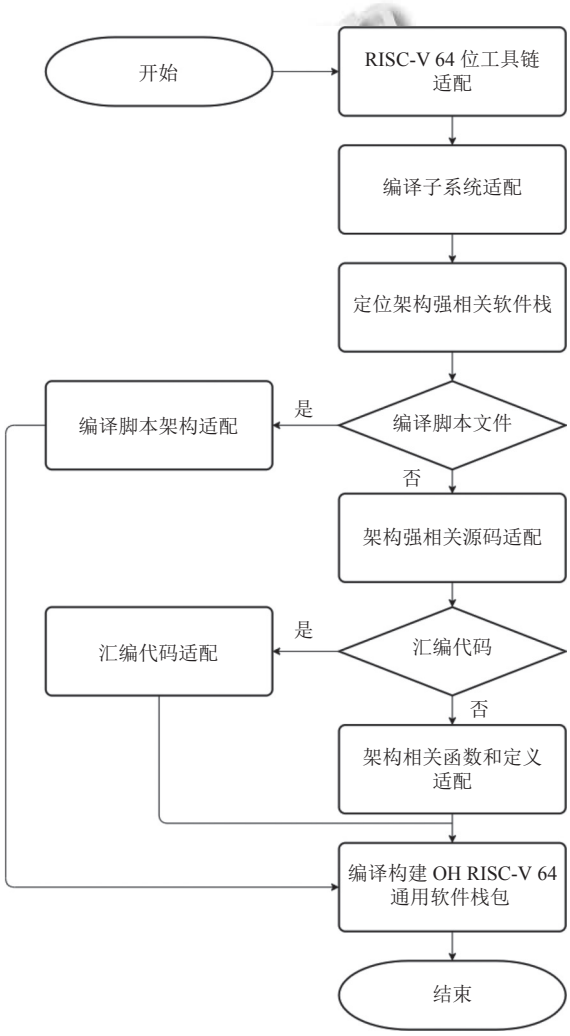

图 2 架构强相关软件栈移植流程图

2.1.1 工具链适配

OH 的源码已完全使用 Clang 来进行系统的整体构建. OH 工具链[8] 主要包括 LLVM 和 libcxx-ndk.llvm 提供了 Clang compiler, lldb, Clang-tidy 等工具, 以及面向不同目标设备的 libc++、Clang_rt、asan、fuzzer 库. 在 LLVM 工具链编译完成后, 再构建 libcxx-ndk, 其包含了对目标平台 ndk 的支持的 libc++库.

目前, LLVM 社区[9] 已支持 RISC-V 指令集架构. 因此, 工具链的适配工作主要是为 RISC-V 添加工具前缀、架构配置以及运行时库相关的支持. 通过 build.py 脚本按照图 3 所示流程完成工具的生成.

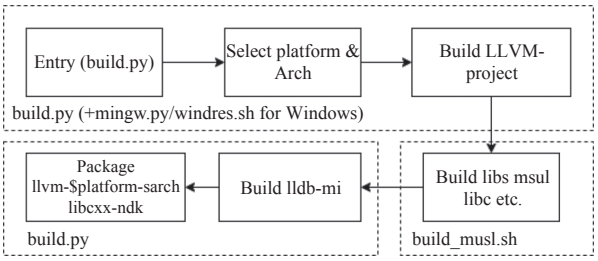

图 3 OH 工具链构建流程

同时, OH 标准系统在面向 RISC-V 架构编译构建时往往会出现工具链导致的兼容性报错, 需要持续解决和优化[10]. 例如, OH 中将配置、图片等文件通过 objcopy 转换成二进制文件, 而在 LLVM12 RISC-V 版本中, 链接器对二进制文件进行了判断, 非体系架构二进制文件会导致报错. 为解决这一编译报错, 我们通过删除此判断, 详细信息见图 4.

```
diff --git a/lld/ELF/Arch/RISCV.cpp b/lld/ELF/Arch/RISCV.cpp
index ca4178b..bfdc786 100644
--- a/lld/ELF/Arch/RISCV.cpp
+++ b/lld/ELF/Arch/RISCV.cpp
@@ -120,7 +120,7 @@ uint32_t RISCV::calcEFlags() const {
     uint32_t eflags = getEFlags(f);
     if (eflags & EF_RISCV_RVC)
       target |= EF_RISCV_RVC;
-
+/*
     if ((eflags & EF_RISCV_FLOAT_ABI) != (target & EF_RISCV_FLOAT_ABI))
       error(toString(f) +
             ": cannot link object files with different floating-point ABI");
@@ -128,6 +128,7 @@ uint32_t RISCV::calcEFlags() const {
     if ((eflags & EF_RISCV_RVE) != (target & EF_RISCV_RVE))
       error(toString(f) +
             ": cannot link object files with different EF_RISCV_RVE");
+*/
   }

   return target;
```

图 4 工具链优化补丁

RISC-V 指令具有模块化的特点. 随着不同平台对扩展指令 (如 V 扩展) 的支持不断优化, 我们可以在 LLVM 中添加指令和寄存器的 tablegen 相关描述, 从而使 OH 标准系统能够得到相应 RISC-V 硬件平台扩展指令的加速优化.

2.1.2 编译框架和构建支持

OH 编译构建子系统采用基于 GN 和 ninja[11,12] 的编译构建框架. 根据产品配置, 可生成对应的镜像包. 编译构建流程如下.

(1) 使用 GN 配置构建目标.

(2) GN 生成 ninja 文件.

(3) 通过运行 ninja 执行编译任务.

OH 编译构建子系统位于 build 目录下, 主要包括编译相关的 Python 脚本、配置项、编译入口 BUILD.gn 配置以及编译打包流程配置, 还有 kits、ndk、notice 版本打包、sa 和 sdk 模板和处理流程、编译工具链配置等. 针对新的 RISC-V 指令集架构, 需要维护函数库路径、编译链接选项、ABI、架构名称字串、架构名称匹配、工具链的路径和参数, 以及与指令集架构相关的编译配置等.

2.1.3 musl C 库适配

基础 C 库主要封装了常用内存操作函数、内核提供的系统调用和数据结构等, 为用户态程序提供了标准化的接口. OH 基础 C 库采用了 musl[13], 它是一个为 Linux 基本系统实现的轻量级、快速、简单、免费、标准兼容和安全的标准库. musl 已支持 RISC-V 指令集. 在 OH 的 GN 编译脚本中, 首要工作是添加 RISC-V 架构选项和架构编译配置. 同时, 为了与 ARM 架构功能对齐, 需要参考 OH musl 已支持的 ARM 架构, 对部分汇编源码和脚本进行 RISC-V 架构适配, 解决由于指令集导致的兼容性问题. 具体代码示例详见图 5.

图 5 musl 适配 OH RISC-V 代码补丁示例

2.1.4 ARK 运行时支持

方舟编译器 (ArkCompiler)[14,15] 是一个统一的编译运行时平台, 旨在支持多种编程语言和多种芯片平台的联合编译和运行. ArkCompiler 主要包括编译工具链和运行时两个部分.

ArkCompiler 的编译工具链以 ArkTS/TS/JS 源码作为输入, 将其编译生成为 abc (ArkCompiler Bytecode, 即方舟字节码) 文件.

ArkCompiler 运行时直接运行字节码文件, 实现对

应语言规范的语义逻辑.

该子系统包含 ets_runtime (ArkTS 运行时组件)、runtime_core (运行时公共组件)、ets_frontend (ArkTS 语言的前端工具) 和 toolchain (ArkTS 工具链).

移植 RISC-V 的关键在于与体系架构相关的运行时公共组件和 ArkTS 工具链. 目前, 官方 ArkTS 工具链已经支持 RISC-V 架构. 因此, 主要的移植工作是完成运行时公共组件中 RISC-V 指令集的汇编功能和 RISC-V 架构的编译配置. 此外, 还需要为 ArkTS 运行时组件添加 RISC-V 架构编译配置.

2.1.5 其他架构相关组件和三方库

除了上述提到的关键软件栈移植之外, 还需要对部分 OH 组件和依赖的第三方库进行编译配置和少量源码的适配工作. 具体列表如表 1 和表 2.

表 1 RISC-V 移植 OH 修改的三方库

三方库	适配类型
boringssl	RISC-V配置适配
curl	RISC-V配置适配
flutter	RISC-V源码适配
libffi	RISC-V源码适配
libunwind	RISC-V源码适配
openssl	RISC-V源码适配

表 2 RISC-V 移植 OH 修改的子系统组件

子系统组件	适配类型
faultloggerd	RISC-V源码适配
ability/ability_runtime	RISC-V配置适配
ace_engine	RISC-V配置适配
napi	RISC-V配置适配
graphic_2d	RISC-V配置适配
webview	依赖架构强相NWeb.hap包
dsoftbus	RISC-V源码适配
Linux/patches	RISC-V源码适配
productdefine/common	RISC-V配置适配

2.2 基于 QEMU RISC-V 架构的 OH 芯片移植-图形显示模块

OH 标准系统芯片移植主要包括产品配置添加、内核启动、升级、音频 ADM 化、Camera、TP、LCD、WiFi、BT、vibrator、sensor 和图形显示等模块的适配. 图形显示模块作为智能终端操作系统的核心功能, 移植优先级最高. 本文将以 QEMU RISC-V 虚拟化平台为设备主体, 完成 OH 图形显示模块的适配, 实现在 RISC-V 架构平台上运行 OH, 并通过 launcher 启动 OH 桌面.

2.2.1 OH 图形显示模块

OH 显示模块架构如图 6 所示. 图形显示模块主要由平台驱动层、芯片平台适配层和 LCD 器件驱动层 3 部分组成.

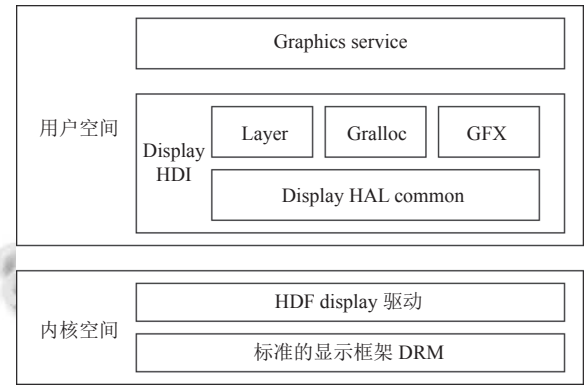

图 6 OH 显示模块架构

(1) Display 平台驱动层

该层通过 HDF (hardware driver framework) 提供的 IOService 数据通道, 与公共 HAL (hardware abstract layer) 对接, 集中接收并处理各类上层调用指令.

(2) SoC 平台驱动适配层

此 SoC 适配层在内核态实现 HDF display 驱动和 SoC 侧驱动 (标准的显示框架) 解耦, 主要完成芯片平台相关的参数配置, 并将平台驱动层的调用传递到器件驱动层.

(3) LCD 器件驱动层

器件驱动层[16] 主要实现与器件自身强相关的驱动适配接口, 例如发送初始化序列、上下电、背光设置等.

基于 display 驱动模型开发 LCD 驱动, 可以借助平台提供的各种能力和接口, 较大程度地降低器件驱动的开发周期和难度, 提升开发效率.

2.2.2 OH 内核移植

(1) 添加显示相关配置

QEMU RISC-V 虚拟化平台[17,18] 官方已支持多种 RISC-V 硬件仿真平台, 例如 sifive_u、shakti_c、virt 等. 本方案将选择 QEMU RISC-V Virt Machine, 该平台是为 RISC-V 软件开发和测试而创建的虚拟平台, 它使用 VirtIO 设备进行网络、存储和其他类型的 IO.

OH 的图形驱动采用 DRM 图形框架, QEMU 可以

使用“virtio-gpu”设备模拟器来模拟 VirtIO GPU 驱动程序, 该驱动程序可以运行在 QEMU 虚拟机中, 并与实际 GPU 硬件交互. Linux 内核已对该驱动提供支持, 需要再内核配置中打开 VirtIO 相关配置项.

```
CONFIG_DRM_VIRTIO_GPU=y
CONFIG_VIRTIO=y
CONFIG_VIRTIO_MENU=y
CONFIG_VIRTIO_PCI=y
CONFIG_VIRTIO_PCI_LEGACY=y
CONFIG_VIRTIO_BALLOON=y
CONFIG_VIRTIO_INPUT=y
CONFIG_VIRTIO_MMIO=y
```

我们通过完成上述内核配置, 在运行于 QEMU RISC-V 虚拟化平台上的 Linux 操作系统中, 用户态即可发现 LCD 设备/dev/dri/card0, 从而实现 OH display HDI 对 QEMU DRM/GPU 的功能的正常访问.

(2) HDF 内核态 (khdf) 补丁

khdf 主要包含驱动程序、驱动配置、驱动资源, 是 OH 驱动开发的基础公共部分. 它完成了驱动的公共逻辑, 资源配置、资源解析以及驱动的加载和初始化功能. 此部分工作主要将 OH HDF 内核补丁移植至 Linux kernel 中.

2.2.3 HAL 层适配

基于 RISC-V 的 OH HDF 驱动开发流程分为两步. 第 1 步是移植 HDF 内核态程序 khdf, 第 2 步是完成 uhdf 的适配, 包含 peripheral 和 framework 的适配.

在 uhdf 适配中, 主要进行用户空间的 HDF 移植. OH 提供了标准的调用流程和使用示例, 位于 Peripheral 目录中, 用于各个驱动外围设备. 通过遵循 HDI (hardware driver interface), 我们能够进行适配、开发和测试, 并保证测试用例通过.

在 display HDI 中, 需要进行 display_device 和 display_gralloc 两部分的适配.

(1) Display device

Display device 模块提供显示设备管理、layer 管理、硬件加速等功能. 在该部分, 我们需要定义 DRM 设备节点.

在 hardware/display/src/display_device/drm_device.cpp 文件中, 我们将 DRM 设备节点名称修改成 QEMU RISC-V 架构对应的“virtio_gpu”节点.

由于 QEMU RISC-V 架构目前不支持硬件合成, 我们需要在 drm_display.cpp 文件中跳过 gfx 的初始化, 并在 hardware/display/src/display_device/hdi_gfx_composition.cpp 文件中修改 set_layers 方法, 使用 CPU 合成显示.

(2) Display gralloc 适配

Gralloc 模块提供显示内存管理功能. 在这部分适配中, 需要根据实际情况修改 hardware/display/src/display_gralloc/display_gralloc_gbm.c 文件中的 DRM 设备节点.

```
const char *g_drmFileNode = “/dev/dri/card0”;
```

2.2.4 OH 图形子系统适配

在完成上述底层内核 DRM 驱动对接和 HAL 层适配后, 我们需要确保图形子系统中 graphic、render、bootanimation、launcher 等服务正常运行[19]. 该部分属于上层组件, 第 2.1 节已完成适配移植工作.

2.3 编译构建并启动 QEMU RISC-V 架构 OH 标准系统

2.3.1 编译构建

面向 OH 标准系统 QEMU RISC-V 架构平台, 我们定义了 qemu_riscv64_virt_linux_system 的产品配置. 在完成上述 OH 主要软件栈 RISC-V 移植和图形显示模块的芯片移植后, 我们可以通过执行./build.sh --product-name qemu_riscv64_virt_linux_system 来编译出 QEMU RISC-V 架构的 OH 镜像, 其中包含内核 Image 和文件系统 ramdisk.img、system.img、updater.img、userdata.img 和 vendor.img 镜像.

2.3.2 QEMU RISC-V 启动 OH

在运行 OH 之前, 我们需要在宿主机安装 QEMU 虚拟机[20]. 本文实验是在 x86-64 位的 Windows 环境, 安装官方 QEMU 7.0 版本 (支持 RISC-V 架构).

QEMU RISC-V 64 虚拟平台硬件参数通过启动命令对设备名称、分配内存、CPU 核数和各设备驱动等进行配置. QEMU RISC-V 架构默认使用 OpenSBI[21] 引导, 并加载内核, 启动 OH 标准系统文件系统中的 init 程序.

成功启动 OH 后, 运行开机动画并进入锁屏界面如图 7 所示, 鼠标解锁后正常进入 OH 界面如图 8 所示.

当前, OH 标准系统 (适配版本 3.2 beta2) 的应用生态相对缺失, 目前仅对原生系统自带的应用, 如设置、计算器、时间等进行测试, 这些系统应用可以正常运行.

在代码量方面, 该工作涉及了 OH 标准系统中 34

个代码仓的适配, 新增了 3 个代码仓, 共计新增 18 000 多行代码.

RISC-V 开发板上的 OH 标准系统移植包括工具链构建、OH 三方库适配、OH 子系统组件适配和芯片移植 4 个部分工作. 前 3 部分的工作具有复用性, 对于新的 RISC-V 开发板移植, 只需完成芯片移植工作, 因此我们提供了 QEMU RISC-V 虚拟平台的芯片移植的参考, 为 RISC-V 开发板在 OH 标准系统上适配提供了 80% 以上的参考依据.

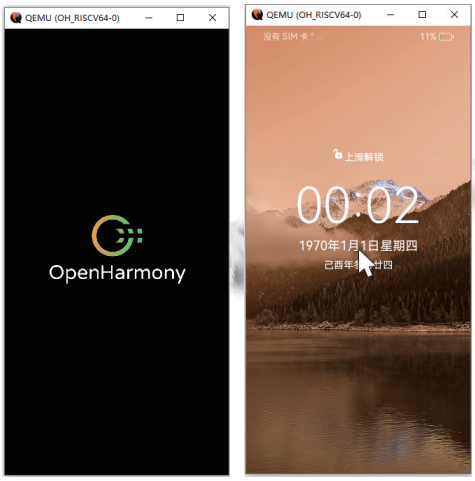

图 7 QEMU RISC-V 架构 OH 开机动画和锁屏界面

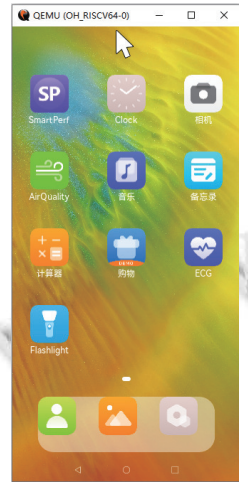

图 8 QEMU RISC-V 架构 OH 桌面

该移植工作的源码和附带的文档是开源的, 读者后续可在聚元 PolyOS 社区[22] 下载.

3 结论与展望

针对 OH 不支持 RISC-V 架构平台的问题, 我们在 QEMU RISC-V 架构上成功移植了 OH 标准系统, 并得出以下经验总结, 希望可以为其他架构迁移工作提供借鉴.

(1) 硬件支持: 在进行架构迁移前, 首先需要了解清楚目标 RISC-V 平台的硬件特性、指令集支持以及设备驱动等关键信息. 确保目标平台与 OH 的要求和功能相匹配, 避免硬件兼容性问题, 是成功迁移的关键基础.

(2) 内核调整: 针对 RISC-V 架构进行必要的内核适配和配置, 确保内核能够正确启动并支持目标平台的硬件特性. 它是整个迁移过程中最核心的一步, 也是保证系统稳定性和性能的关键.

(3) 驱动程序适配: 设备驱动程序的适配是确保目标平台上硬件设备能够与 OH 系统正常通信和工作的关键步骤. 系统需要完整的硬件支持, 驱动程序适配是其中不可或缺的一环.

(4) 解决指令集差异: RISC-V 指令集可能与现有的 OH 所基于的 ARM 架构存在差异. 因此, 在进行移植工作时, 需要仔细解决指令集差异, 进行指令集转换和适配工作, 以确保软件在新的架构上能够正确运行.

(5) 选择合适的 OH 版本: 选择与目标平台相匹配的 OH 版本是保证兼容性的重要因素. OH 标准系统存在多个版本, 为了保证成功迁移, 需要选择与目标平台相匹配的 OH 版本. 这样可以确保内核和应用程序的兼容性, 避免不必要的兼容性问题和冲突.

(6) 应用程序兼容性: 应用程序兼容性对于整个迁移工作非常重要, 确保 OH 应用程序能够在新的 RISC-V 平台上正确运行, 需要重新编译或调整应用程序以适应新的硬件和操作系统环境. 对于核心应用程序, 兼容性测试是必不可少的.

(7) 开源社区合作: 参与开源社区合作是推进 RISC-V 生态和 OH 移植工作的重要手段. 与其他开发者分享经验, 寻求帮助和合作, 同时也可以获得更多有用的经验和资源. 在开源社区中共同探讨和解决问题, 可以大大提高迁移工作的效率和质量.

在进行类似工作时, 重点关注硬件支持、内核调整、驱动程序适配和解决指令集差异等方面, 可以有效地引导开发者完成架构迁移, 提高工作的成功率和效率.

这项工作促进了 RISC-V 软硬生态的发展. 通过提供开源的虚拟化环境, 我们为 RISC-V OH 应用生态提

供了支持, 使开发者能够快速进行 RISC-V 平台的 OH 应用开发和测试. 此外, 我们为不断发布的 RISC-V 开发板提供 OH 芯片移植参考, 大大提高了 RISC-V 开发板在 OH 的适配效率, 丰富了 OH 的硬件生态, 并推动了 OH 开源社区的发展.

目前 QEMU RISC-V 架构已初步完成 OH 图形模块的移植. 接下来, 我们将继续推进音频、网络、键鼠、摄像头等驱动模块的芯片移植工作, 为开发者提供更全面的驱动设备支持, 提升用户的交互体验. 同时, 我们还将对 QEMU RISC-V 架构的功能和稳定性进行验证测试, 并结合与 ARM 架构相比的性能差异点进行优化[23], 例如指令集的使用效率、缓存和内存访问的优化等. 随着 OH 上层 UI 组件的不断迭代, 应用生态将逐渐丰富. 我们将对各类应用进行适配测试, 以验证系统移植的正确性和鲁棒性.

参考文献

1 Waterman A, Lee Y, Patterson DA, *et al*. The RISC-V instruction set manual, volume I: Base user-level ISA. Technical Report, Berkeley: University of California, 2011. 116.

2 Greengard S. Will RISC-V revolutionize computing? Communications of the ACM, 2020, 63(5): 30–32. [doi: 10.1145/3386377]

3 温婷. 阿里 RISC-V 拿到商业化“入场券”. 上海证券报, 2023-03-03(07). [doi: 10.28719/n.cnki.nshzj.2023.000877]

4 Singhal SP, Sridevi M, Narayanan NS, *et al*. Porting of eChronos RTOS on RISC-V architecture. Proceedings of the 2021 International Conference on Advanced Communication and Computational Technology. Singapore: Springer, 2021. 1269–1279.

5 Wang WZ, Liu XD, Yu J, *et al*. The design and building of openKylin on RISC-V architecture. Proceedings of the 15th International Conference on Advanced Computer Theory and Engineering (ICACTE). Hangzhou: IEEE, 2022. 88–91.

6 种丹丹. 基于 RISC-V 的开源芯片生态发展现状及未来机遇. 中国集成电路, 2021, 30(8): 25–30. [doi: 10.3969/j.issn.1681-5289.2021.08.005]

7 OpenHarmony. https://gitee.com/openharmony. [2023-03-12].

8 OpenHarmony LLVM-project. https://gitee.com/openharmony/third_party_llvm-project. [2023-03-12].

9 Chris L. RISC-V LLVM. https://github.com/sifive/riscv-llvm. [2023-03-12].

10 Krill P. LLVM 12 arrives with x86, AArch optimizations. https://www.infoworld.com/article/3615511/llvm-12-arrives-with-x86-aarch-optimizations.html. (2021-04-21)[2023-03-15].

11 王奥维, 王煜, 傅杰辉, 等. 基于多源数据融合算法的华为鸿蒙智慧农业系统. 智慧农业导刊, 2023, 3(9): 1–4. [doi: 10.20028/j.zhnydk.2023.09.001]

12 ninja. https://github.com/ninja-build/ninja. [2023-03-20].

13 musl libc. http://www.musl-libc.org/. [2023-03-20].

14 史宁宁. 华为方舟编译器之美: 基于开源代码的架构分析与实现. 北京: 清华大学出版社, 2020.

15 方舟运行时子系统. https://gitee.com/openharmony/docs/blob/master/zh-cn/readme/ARK-Runtime-Subsystem-zh.md. [2023-04-10].

16 Lawall J, Palinski D, Gnirke L, *et al*. Fast and precise retrieval of forward and back porting information for Linux device drivers. Proceedings of the 2017 USENIX Conference on USENIX Annual Technical Conference. Santa Clara: USENIX Association, 2017. 15–26.

17 Pieper P, Herdt H, Drechsler R. Advanced embedded system modeling and simulation in an open source RISC-V virtual prototype. Journal of Low Power Electronics and Applications, 2022, 12(4): 52. [doi: 10.3390/jlpea12040052]

18 Zabołotny WM. QEMU-based hardware/software co-development for DAQ systems. Journal of Instrumentation, 2022, 17: C04004. [doi: 10.1088/1748-0221/17/04/C04004]

19 董庆平. 基于 R2R 平台的 Android 系统移植及图形系统研发 [硕士学位论文]. 武汉: 武汉理工大学, 2014.

20 汪辰. 在 QEMU 上运行 RISC-V 64 位版本的 Linux. https://zhuanlan.zhihu.com/p/258394849. [2023-04-20].

21 OpenSBI. RISC-V. https://www.oschina.net/p/opensbi?hmsr=aladdin1e1. [2023-05-08].

22 聚元 PolyOS. https://gitee.com/riscv-raios. [2023-04-15].

23 刘畅, 武延军, 吴敬征, 等. RISC-V 指令集架构研究综述. 软件学报, 2021, 32(12): 3992–4024. [doi: 10.13328/j.cnki.jos.006490]

(校对责编: 孙君艳)