·体系结构与软件技术· 文章编号：1000-3428(2017)08-0095-06 文献标志码：A 中图分类号：TP391

基于 Chromium 的渲染进程轻量化隔离方法

肖伟民[1,2]，邓浩江[1]，胡琳琳[1]，郭志川[1]

（1. 中国科学院声学研究所 国家网络新媒体工程技术研究中心，北京 100190；2. 中国科学院大学，北京 100190）

摘 要： 为实现浏览器对 Web 应用的高效管理和安全隔离，提出一种新的渲染进程轻量化隔离方法。研究并分析 Chromium 多进程机制及管理策略与 Docker 容器虚拟化技术。设计 Chromium 与 Docker 相结合的 ZygoteDocker 方案，将渲染进程模块从浏览器核心模块中分离，同时精简浏览器功能实现浏览器的轻量化。实验结果表明，该渲染进程模块在容器内实现了隔离，轻量化结果较为明显。

关键词： Chromium 浏览器；Docker 容器；虚拟化；渲染进程；轻量化；隔离

中文引用格式： 肖伟民，邓浩江，胡琳琳，等. 基于 Chromium 的渲染进程轻量化隔离方法[J]. 计算机工程，2017，43(8)：95-100.

英文引用格式： Xiao Weimin，Deng Haojiang，Hu Linlin，et al. Lightweight Isolation Method of Rendering Process Based on Chromium[J]. Computer Engineering，2017，43(8)：95-100.

Lightweight Isolation Method of Rendering Process Based on Chromium

XIAO Weimin[1,2]，DENG Haojiang[1]，HU Linlin[1]，GUO Zhichuan[1]

（1. National Network New Media Engineering Research Center，
Institute of Acoustics，Chinese Academy of Sciences，Beijing 100190，China；
2. University of Chinese Academy of Sciences，Beijing 100190，China）

【Abstract】 To implement efficient management and secure isolation of Web application in browser，this paper proposes the lightweight isolation method of rendering process based on Chromium. It researches and analyzes multi-process mechanism and management of Chromium，as well as virtualization of Docker container，designs the solution of ZygoteDocker which is the combination of Chromium and Docker，separates the rendering process module from the browser content module，and implements lightweight browser by simplifying the browser functions. Experimental result shows that the rendering process module implements the isolation in the container，with obvious lightweight result.

【Key words】 Chromium browser；Docker container；virtualization；rendering process；lightweight；isolation

DOI： 10.3969/j.issn.1000-3428.2017.08.016

0 概述

随着互联网技术的高速发展，嵌入式系统[1]越来越智能化，用户对嵌入式设备需求的日渐提高，嵌入式终端设备[2]也越来越多。在嵌入式终端运行着众多应用，用户的部分隐私数据会存储在系统中，如账号、密码等。在传统的嵌入式系统中，木马、恶意软件和病毒很容易侵入并盗取用户的隐私数据，安全性存在很大问题，因此，对应用进行隔离[3-4]运行是一种有效的安全策略。

此外，智能化的嵌入式设备在性能上虽然得到了提升，但是对于用户需求的高分辨率视频、复杂的图形处理等，需要更高效的资源管理[5]技术，因此，对应用进行统一、高效的管理也是必要的。并且由于嵌入式设备处理能力有限，需要对 Web 运行环境[6]进行轻量化处理，使之能够占用更少的系统资源，获得更高的执行效率。

容器虚拟化[7-8]技术能有效解决上述关于应用的管理和安全问题，其中以 Docker[9]为代表的新一代容器技术是研究的热点。将应用的隔离与 Docker 容器技术相结合，可极大满足应用的高效管理与安全性的要求。

基金项目： 国家“863”计划项目“动态媒体的多元适配与耦合技术研究”(2015AA015802)。

作者简介： 肖伟民(1989—)，男，博士研究生，主研方向为新媒体技术；邓浩江，研究员、博士；胡琳琳，副研究员、硕士；郭志川，副研究员、博士。

收稿日期： 2016-04-18 **修回日期：** 2016-07-08 **E-mail：** xiaowm@dsp.ac.cn

针对嵌入式终端的轻量化和安全性的要求,本文提出一种基于 Chromium 的渲染进程[10]轻量化隔离方法,将渲染进程模块从浏览器核心模块分离,运行在容器内部进行隔离管理,同时精简浏览器功能以达到浏览器轻量化的目的。

1 Chromium 的多进程机制

在单进程浏览器中,浏览器或插件的错误可能导致整个浏览器和所有的标签页面崩溃退出,这种情况对用户当前的工作或娱乐活动都是非常不好的体验。

将不同的标签页面等放置于不同的进程中以避免相互之间的影响,Chromium 的鲁棒性得到了极大的提高,一个页面的错误只会导致当前页面的崩溃,整个浏览器的使用仍然不会受到太大的影响。

1.1 Chromium 的多进程架构

在 Chromium 多进程架构中[11],对于浏览器的每一个标签页使用独立的进程来运行,对不同进程之间的访问和进程到系统的访问也有严格的限制,通过这些措施,可以保护浏览器不同标签页面之间的内存资源不受干扰,使得浏览器更加稳定。如图 1 所示,Chromium 将运行浏览器 UI 和管理功能的主进程称为“Browser Process”,而运行标签页的渲染进程称为“Render Process”。

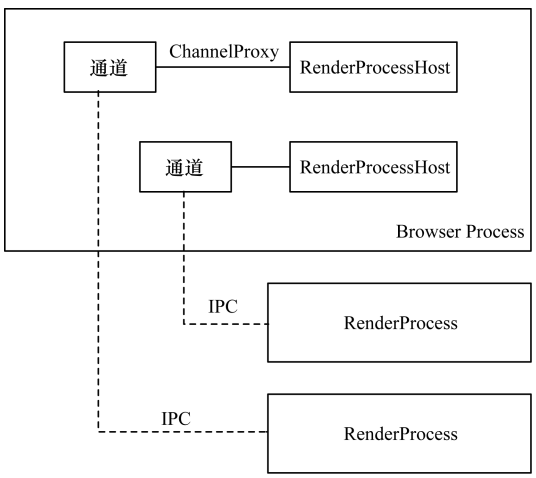
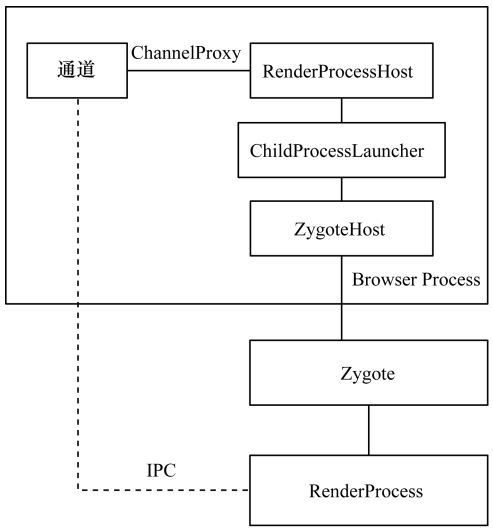


图 1 Chromium 的多进程架构

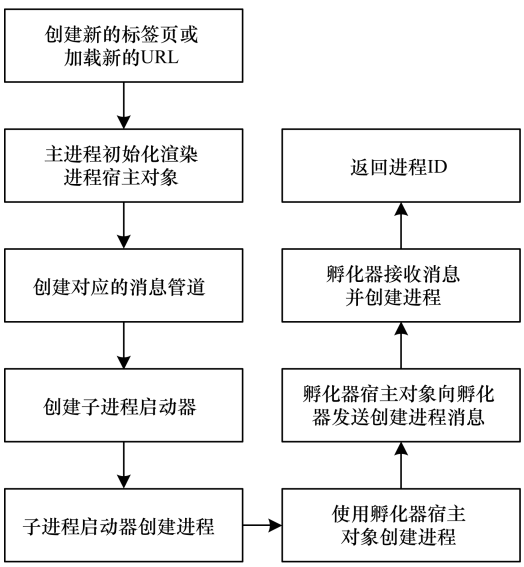
浏览器主进程对渲染进程的管理通过一一对应的渲染进程宿主对象(RenderProcessHost)和渲染进程对象(RenderProcess)进行。对于每一个渲染进程,有一个全局的渲染进程对象,用于和主进程进行通信和维护全局的状态;主进程有一个相应的渲染进程宿主对象,用于管理主进程状态和与渲染进程进行通信。主进程和渲染进程之间通过消息管道进行进程间通信(Inter Process Communication,IPC)[12]。

1.2 Chromium 主进程对渲染进程的管理

Chromium 主进程对渲染进程的管理,包含渲染进程的创建、优先级处理、关闭 3 个部分。图 2 ~ 图 4分别为 Chromium 主进程对渲染进程进行管理的框架和流程,通过分析对比可以得出,进程的管理不是通过统一的孵化器宿主对象(ZygoteHost)和孵化器(Zygote)进行的。孵化器主要完成了进程的创建工作,是连接主进程和渲染进程的主体;进程的关闭并未在孵化器内进行,孵化器只是进行关闭后的处理工作;进程的优先级调整则未通过孵化器。

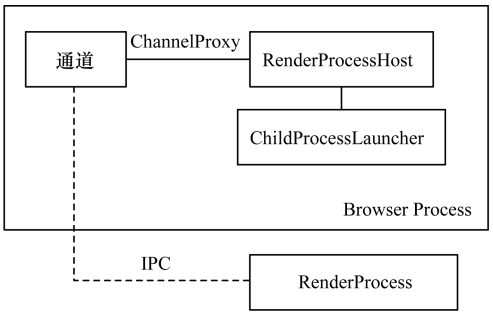


(a)渲染进程创建框架

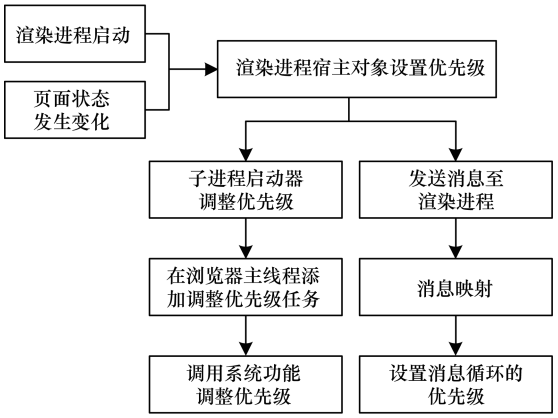


(b)渲染进程创建流程

图 2 渲染进程创建的框架及流程

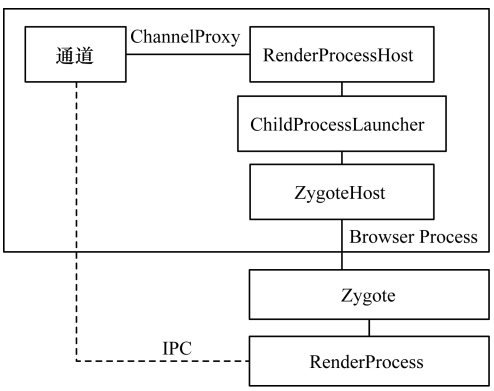


(a)渲染进程调整优先级框架

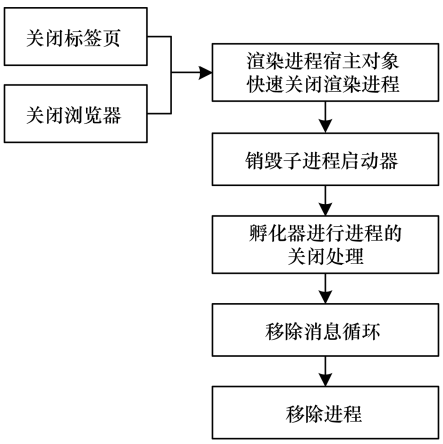


(b)渲染进程调整优先级流程

图 3　渲染进程调整的优先级框架及流程



(a)渲染进程关闭框架



(b)渲染进程关闭流程

图 4　渲染进程关闭的框架及流程

主进程对渲染进程的不同管理流程不统一,对于应用的统一高效管理和安全隔离十分不利。针对此问题,本文设计了孵化器的功能,与容器技术相结合,所有流程均通过孵化器来进行应用的管理。

2　Docker 容器技术

Docker 是基于 Linux 的多项开源技术提供的高效、敏捷和轻量级的容器方案,并且支持在多种主流平台和本地系统上部署,可以实现“一次封装,到处运行”。

2.1　容器技术

Docker 引擎的基础是 Linux 容器技术(LXC),在 LXC 的基础上,Docker 进一步优化了容器的使用体验。Docker 提供了各种容器管理工具(如分发、版本、移植等)让用户无需关注底层的操作,可以更简单地管理和使用容器。Docker 架构如图 5 所示,最底层的硬件设备上运行着 Linux 内核,Docker 引擎基于 Linux 内核技术管理着最顶层的容器,每个容器都是互相隔离的,都运行一个用户的应用程序或进程。

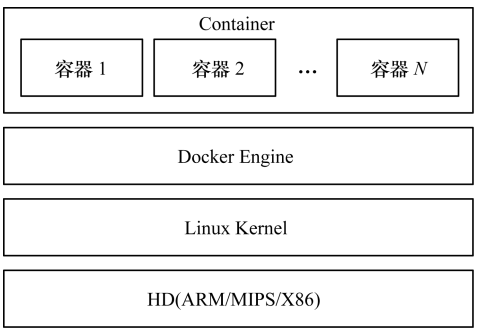


图 5　Docker 架构

2.2　Docker 的优势

以 Docker 为代表的容器虚拟化技术具有以下优势:轻量级虚拟化,秒级启动,秒级停止,空间资源占用较少,进程级别的隔离,高效的资源利用率以应用为中心、自动构建等。

以上的优点使得容器虚拟化技术成为研究的热点内容,将 Docker 与 Chromium 的进程隔离相结合,以实现 Web 应用轻量化隔离,能够更加高效地管理资源和轻量化,在嵌入式终端取得更加安全、更加快速的用户体验。

3　与 Docker 结合的轻量化隔离方案

本文基于 Chromium 提出与 Docker 容器技术相结合的渲染进程轻量化隔离方案,以实现 Web 应用的安全运行与高效管理,方案架构如图 6 所示。

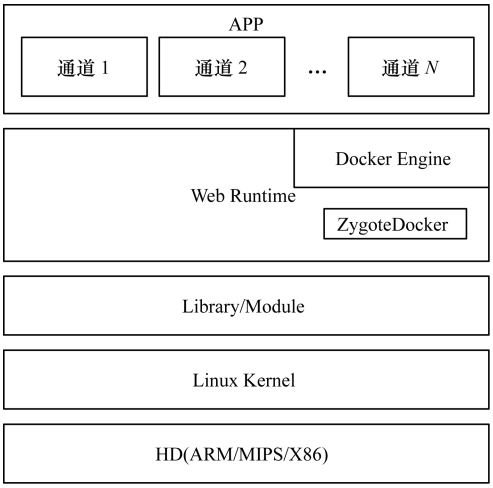


图 6 Chromium + Docker 轻量化隔离架构

该方案具体过程如下：

1）Web 应用独立运行于容器内。

2）Docker 引擎的主要作用是为 Web 运行环境提供接口和服务，将每个 Web 应用通过容器技术进行隔离，并提供完整容器生命周期管理，包括启动、运行、暂停、恢复、停止、迁移[13-14]。

3）在 Web 运行环境中添加重新设计的孵化器与 Docker 客户端相结合的 ZygoteDocker，用于与 Docker 引擎交互，管理容器和获取容器状态信息。

该设计方案的特点在于与容器技术结合，通过容器管理渲染进程；所有对渲染进程的管理均通过重新设计的 ZygoteDocker 与 Docker 引擎进行交互、由 Docker 引擎对运行着渲染进程的容器进行管理。

3.1 与 Docker 结合的 ZygoteDocker 技术方案

基于上述 Chromium 与 Docker 结合的轻量化隔离方案，结合 1.2 节所分析对比的结果，本文提出了采用 Chromium 孵化器和 Docker 客户端相结合为 ZygoteDocker 的方法。

技术方案设计如图 7 所示，其特点如下：

1）主进程运行 ZygoteDocker 宿主对象，负责与 ZygoteDocker 客户端进行交互，当主进程对渲染进程进行管理或通信时，将处理指令层层传递至 ZygoteDocker 宿主对象，由 ZygoteDocker 宿主对象将处理指令发送到 ZygoteDocker 客户端。

2）ZygoteDocker 客户端接收到来自主进程的处理指令后，触发相应处理机制，与 Docker 引擎进行交互，将管理或通信等信息交于 Docker 引擎。

3）Docker 引擎依据 ZygoteDocker 客户端的处理请求，对各个容器进行管理与传递消息。

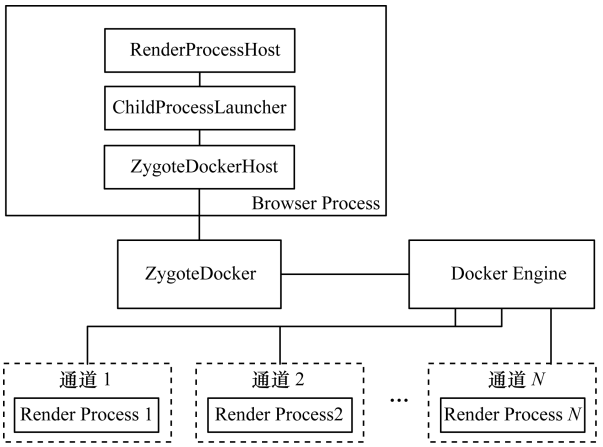


图 7 ZygoteDocker 技术方案

该方案与 Chromium 原有管理机制的区别在于 Docker 引擎的引入，由 Docker 引擎对容器进行管理；渲染进程运行于容器内，只接受 Docker 引擎管理；重新设计的 ZygoteDocker，负责与 Docker 引擎的交互；主进程对渲染进程的管理和通信均通过 ZygoteDocker。

3.2 ZygoteDocker 功能设计

ZygoteDocker 功能设计如图 8 所示，ZygoteDocker 客户端的基本功能包括进程的管理（进程创建、调整进程的后台和前台状态、进程销毁）、进程的通信（发送或接收进程消息）、进程状态的获取等。ZygoteDocker 客户端统一处理来自主进程 ZygoteDocker 宿主对象的请求和消息，并与 Docker 引擎进行交互。

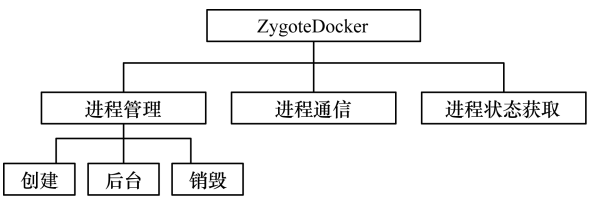


图 8 ZygoteDocker 客户端的功能设计

3.3 方案改进

本文提出的采用 Chromium 孵化器和 Docker 客户端相结合为 ZygoteDocker 的方法，相对于 Chromium 原有的设计方案，其改进之处在于改变 Chromium 原有的主进程直接管理渲染进程的方法，与容器技术结合，通过容器管理渲染进程；改变原有不统一的管理流程，所有对渲染进程的管理均通过重新设计的 ZygoteDocker 与 Docker 引擎进行交互，由 Docker 引擎对运行着渲染进程的容器进行管理，依托容器技术可以提供更加高效和全面的资源管理、安全隔离策略。

4 系统实现难点与解决方案

4.1 渲染进程分离

由于 Chromium 原有的设计为渲染进程模块（content_renderer）处于核心模块（content）内，且与内

部其他模块紧耦合,若将渲染进程放置于 Docker 容器内运行,则需要将渲染进程模块与核心模块进行解耦分离,如图 9 所示,图上方为 Chromium 原有设计及依赖关系,模块之间的耦合度很高,图下方为渲染进程模块分离方案设计,其关键点在于:分离耦合度模块;将核心模块(content)内的通用模块(content_common)和子模块(content_child)独立;将渲染模块(content_renderer)从核心模块内分离;模块依赖重定向。

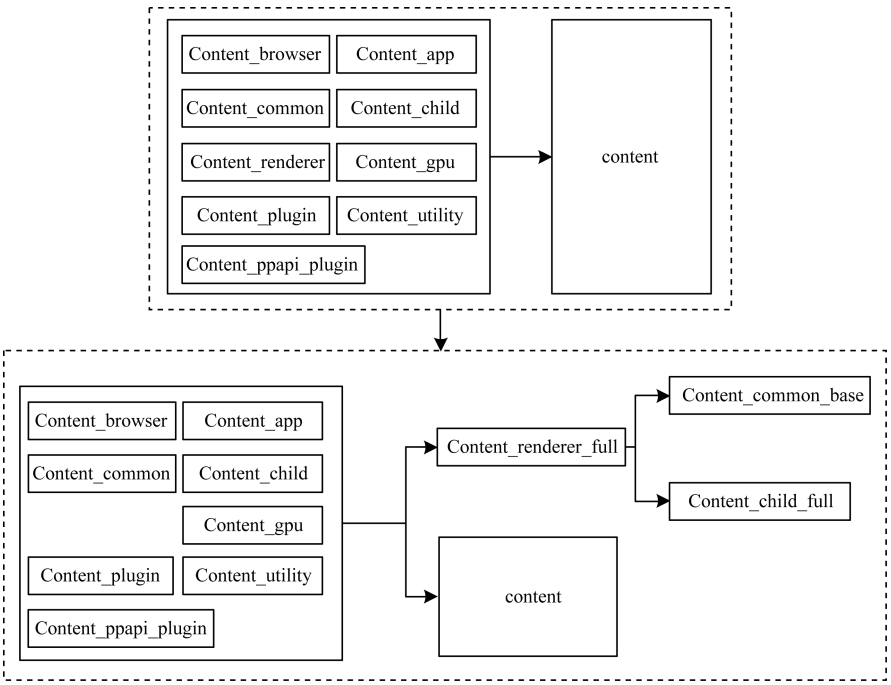


图 9　渲染进程分离

完成渲染进程模块的分离后,将渲染进程模块设计为容器内的可运行对象,使之能在容器内独立运行,其关键点在于:创建独立的可运行对象;模块依赖的分析与定向;创建渲染进程运行入口。

4.2　容器管理引擎模块

将 Chromium 渲染进程使用 Docker 进行以容器为单位的管理,需要独立实现中间件[15]模块:容器管理引擎模块。Docker 通过该模块可为 Chromium 提供容器的管理功能。

容器管理引擎模块提供的功能如下:进程的管理(进程创建、调整进程的状态、进程销毁),进程的通信(发送或接收进程消息),进程状态的获取等。

容器管理引擎模块实现的难点包括:

1) Chromium 与 Docker 之间的管理功能实现。由于 Chromium 与 Docker 为分别独立的,将两者结合需要解决不同编程语言、功能需求不同等问题,实现中间件模块来解决相关问题是较为有效的方法。

2) 容器 ID 与进程 ID 之间的哈希[16]映射。Chromium 创建一个新的进程需要获取新的进程号来对渲染进程进行区分,而容器管理引擎创建新的容器则是得到一个容器 ID,将两者进行唯一对应关系的映射是将两者结合的必要方法,本文在系统实现方面使用哈希映射的方法,将创建的容器 ID 与进程号进行映射,以达到 Chromium 可通过 Docker 管理容器的目的。

4.3　Chromium 与容器通信

Chromium 与容器进行通信是系统实现需要解决的一大难点,由于 Chromium 使用未命名的 Socketpair 进行通信,在进程隔离的容器内部,Socketpair 无法找到通信对象,这是无法正确建立连接并进行通信的。

对于以上技术难点,其解决方案为:在与 Docker 进行结合的隔离容器内,需要将整个通信机制改进为命名的 Unix Domain Socket 进行通信,可以在隔离的容器之间建立一条通信管道进行正确的通信。

5　系统实现与测试分析

本文方法在 Chromium Version:43.0.2338.0 源码基础上进行实现,采用 GYP 编译系统进行交叉编译得到动态库和可执行文件,Docker 源码版本为 V1.9.0-dev,编译环境为 Ubuntu 14.04 LTS 服务器。测试环境为嵌入式 HiSi STB V300 机顶盒。完成了渲染进程的分离及轻量化,分离的轻量化渲染进程在 Docker 引擎中完成运行,且进行了轻量化结果对比测试,测试结果如表 1 所示。

表 1 部分动态库轻量化结果对比 MB

动态库	原有大小	轻量化大小
libcontent. so	229.00	14.20
libevents. so	1.21	0.12
libipc. so	1.99	0.18
libbase. so	14.50	1.30

6 结束语

随着互联网技术的高速发展,嵌入式系统智能化程度越来越高,智能终端对于应用的安全性提出了更高的要求。随着智能终端对复杂计算处理的要求越来越高,需要对应用进行更加高效的管理。为实现浏览器对 Web 应用的高效管理和安全隔离,本文提出一种基于 Chromium 的渲染进程轻量化隔离方法,分析了 Chromium 的多进程架构以及主进程对渲染进程的管理机制,包括进程的创建、进程的优先级处理、进程的关闭以及进程间通信等。介绍了 Docker 虚拟化容器技术及优势,给出与 Docker 相结合的渲染进程轻量化隔离方案,以及 ZygoteDocker 的技术方案设计、功能设计与渲染进程分离的设计,并基于此进行系统的初步实现和测试,结果表明,该方法渲染进程模块在容器内实现了隔离,轻量化结果较为明显。

参考文献

[1] Malinowski A, Yu H. Comparison of Embedded System Design for Industrial Applications[J]. IEEE Transactions on Industrial Informatics, 2011, 7(2): 244-254.

[2] Xie X W, Mao H, Xie X W, et al. Mobile Intelligent Terminal Equipment Learning Software Design and Development[C]//Proceedings of International Conference on Intelligent Transportation, Big Data & Smart City. [S.l.]: IEEE Computer Society, 2015: 515-518.

[3] 赵 波,夏忠林,安 杨,等. 用于虚拟化环境的进程隔离方法研究与实现[J]. 华中科技大学学报(自然科学版), 2014(11): 74-79.

[4] 李 瑜,赵 勇,梁 鹏. 面向服务进程的用户权限隔离模型[J]. 计算机工程, 2011, 37(23): 141-143.

[5] Huang D, He B, Miao C. A Survey of Resource Management in Multi-tier Web Applications[J]. IEEE Communications Surveys & Tutorials, 2014, 16(3): 1574-1590.

[6] 刘秀秀,潘 梁,郭志川,等. 基于 Web 运行环境的 Android 原生应用管理研究[J]. 网络新媒体技术, 2015, 4(4): 35-40.

[7] Kristinesønn S R. A Study of Linux Containers and Their Ability to Quickly Offer Scalability for Web Services[D]. Oslo, Kongeriket Norge: University of Oslo, 2015.

[8] Chen W, Xu L, Li G, et al. A Lightweight Virtualization Solution for Android Devices[J]. IEEE Transactions on Computers, 2015(1): 2741-2751.

[9] Bernstein D. Containers and Cloud: From LXC to Docker to Kubernetes[J]. IEEE Cloud Computing, 2014, 1(3): 81-84.

[10] Reis C. Process Models[EB/OL]. (2013-01-15). https://www.chromium.org/developers/design-documents/process-models.

[11] Wilson B, Goodger B. Multi-process Architecture[EB/OL]. (2015-04-30). https://www.chromium.org/developers/design-documents/multi-process-architecture.

[12] Harris W. Inter-process Communication[EB/OL]. (2015-11-13). https://www.chromium.org/developers/design-documents/inter-process-communication.

[13] Nider J, Rapoport M. Cross-ISA Container Migration[C]// Proceedings of ACM International on Systems and Storage Conference. New York, USA: ACM Press, 2016: 259-264.

[14] Hacker T J, Romero F, Nielsen J J. Secure Live Migration of Parallel Applications Using Container-based Virtual Machines[J]. International Journal of Space-based and Situated Computing, 2012, 2(1): 45-57.

[15] Vinoski S. Where is Iddleware[J]. IEEE Internet Computing, 2002, 6(2): 83-85.

[16] Maurer W D, Lewis T G. Hash Table Methods[J]. ACM Computing Surveys, 1975, 7(1): 5-19.

编辑 索书志

(上接第 94 页)

[8] Ramchandani C. Analysis of Asynchronous Concurrent Systems by Timed Petri Nets[D]. Cambridge, USA: Massachusetts Institute of Technology, 1973.

[9] 刘昊知,胥布工,高福荣. 基于时序模型迁移的间歇过程监测建模方法[J]. 控制与决策, 2015, 30(10): 1885-1889.

[10] 黄姝娟,朱怡安,李兵哲,等. 具有依赖关系的周期任务实时调度方法[J]. 计算机学报, 2015, 38(5): 999-1006.

[11] Oh J, Kyoung-Don K. A Predictive-reactive Method for Improving the Robustness of Real-time Data Services[J]. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(5): 974-986.

[12] Zhou Yan, Kang K D. Deadline Assignment and Feedback Control for Differentiated Real-time Data Services[J]. IEEE Transactions on Knowledge and Data Engineering, 2015, 27(12): 3245-3257.

[13] Alur R, Dill D L. A Theory of Timed Automata[J]. Theoretical Computer Science, 1994, 126(2): 183-235.

[14] 赵 罡,何 锋,徐亚军,等. 时间触发总线流量调度机制及其实时性分析[J]. 计算机工程, 2015, 41(10): 59-65.

[15] 张彬连,徐洪智. 基于随机任务的可靠性约束与节能调度算法[J]. 计算机工程, 2015, 41(8): 1-5.

[16] Song Han, Lam K, Wang Jiantao. Adaptive Co-scheduling for Periodic Application and Update Transactions in Real-time Database Systems[J]. Journal of Systems and Software, 2012, 85(8): 1729-1743.

[17] 黄国兵,李瑞玲,李华丽,等. μC/OS-II 任务优先级调度算法分析与改进[J]. 计算机工程, 2015, 41(8): 52-54, 60.

[18] Emesto W, Lothar T. Real-Time Calculus (RTC) Toolbox[EB/OL]. [2016-03-20]. http://www.map.ethz.ch/Rtctoolbox.

编辑 金胡考