计算机科学
COMPUTER SCIENCE

基于多核异构操作系统的动态冗余可靠机制研究

何瑞琦，张凯龙，吴金飞，于强，张家铭

引用本文

何瑞琦，张凯龙，吴金飞，于强，张家铭. 基于多核异构操作系统的动态冗余可靠机制研究[J]. 计算机科学, 2025, 52(4): 33-39.

HE Ruiqi, ZHANG Kailong, WU Jinfei, YU Qiang, ZHANG Jiaming. Research on Dynamic Redundancy Reliability Mechanisms Based on Multi-core Heterogeneous Operating Systems [J]. Computer Science, 2025, 52(4): 33-39.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.241100020

基于多核异构操作系统的动态冗余可靠机制研究

何瑞琦[1]　张凯龙[1]　吴金飞[1]　于　强[2]　张家铭[1]
1 西北工业大学软件学院　西安 710129
2 西北工业大学计算机学院　西安 710129
(2543267094@qq.com)

摘　要　针对当前嵌入式系统的混合部署需求和功能安全需求,提出了一种动态异构冗余的操作系统架构 DHR-OS。面向混合部署需求,该架构设计了异构操作系统的混合部署模式,即在多核 CPU 上以 Linux 为主操作系统,动态部署 RTOS 从操作系统镜像。同时,为了操作系统间的协同工作,利用 OpenAMP(Open Asymmetric Multi-Processing)实现了主从操作系统间的通信,并基于 OpenAMP 进一步实现了设备驱动的时分复用、远程过程调用(Remote Procedure Call,RPC)、中断转发路由机制。面向功能安全需求,该架构设计了一套调度、分发、裁决为一体的关键任务安全执行机制。具体地,Linux 操作系统事先对 RTOS 核心做池化处理,当执行关键任务时,从 RTOS 内核池中调度若干 RTOS 核心作为任务执行环境,在 Linux 侧的裁决器基于加权投票的分布式共识算法,处理 RTOS 核心任务返回的结果。通过上述设计增强了系统的灵活性和抗攻击能力。该工作为嵌入式系统的混合部署和功能安全需求提供了一种新的系统架构解决方案,具有一定创新性和实用价值。

关键词:异构动态冗余;混合部署;功能安全;分布式共识
中图分类号　TP316

Research on Dynamic Redundancy Reliability Mechanisms Based on Multi-core Heterogeneous Operating Systems

HE Ruiqi[1],ZHANG Kailong[1],WU Jinfei[1],YU Qiang[2] and ZHANG Jiaming[1]
1 School of Software,Northwestern Polytechnical University,Xi'an 710129,China
2 School of Computer Science,Northwestern Polytechnical University,Xi'an 710129,China

Abstract　In response to the hybrid deployment requirements and functional safety needs of current embedded systems,this paper proposes a dynamic heterogeneous redundant operating system architecture,DHR-OS. Designed for hybrid deployment,the architecture features a mixed deployment model of heterogeneous operating systems,where Linux serves as the primary operating system on a multi-core CPU,while RTOS is dynamically deployed from the operating system image. To facilitate collaboration between operating systems,communication between the master and slave operating systems is implemented using OpenAMP. Furthermore,based on OpenAMP,mechanisms for time-division multiplexing of device drivers,remote RPC calls,and interrupt forwarding routing are established. To address functional safety requirements,the architecture includes a critical task safety execution mechanism that integrates scheduling,dispatching,and adjudication. Specifically,the Linux operating system pre-processes a pool of RTOS cores. When executing critical tasks,several RTOS cores are scheduled from this pool to serve as the task execution environment. The adjudicator on the Linux side processes the results returned by the RTOS core tasks using a distributed consensus algorithm based on weighted voting. This design enhances the system's flexibility and resilience against attacks,providing a novel architectural solution to the hybrid deployment and functional safety needs of embedded systems,with significant innovation and practical value.

Keywords　Dynamic heterogeneous redundancy,Hybrid deployment,Functional safety,Distributed consensus

到稿日期:2024-11-04　返修日期:2025-01-31
基金项目:国家自然科学基金(61972318);陕西省重点研发计划(2023-GHZD-47);上海航天技术研究院产学研合作基金项目(SAST2024-007)
This work was supported by the National Natural Science Foundation of China (61972318),Key R&D Program of Shaanxi Province(2023-GHZD-47) and Shanghai Aerospace Technology Research Institute Industry University Research Cooperation Fund Project(SAST2024-007).
通信作者:张凯龙(kl.zhang@nwpu.edu.cn)

1 引言

随着信息物理融合系统[1]、智能物联网[2]等高性能嵌入式应用场景的快速发展与广泛应用,嵌入式系统整体逐渐呈现出通用化、智能化、协同化的新趋势,其内部资源也展现出异构化和多态化特征。以汽车电子行业为例,传统汽车座舱的电子控制单元[3]生态系统主要呈现为碎片化,堆叠式电子控制单元之间的信息无法有效交互。随着车-路-云概念的兴起,汽车内各个部件的交互以及汽车与路云的协同变得更加频繁。面对分散式控制单元通信链路长、交互度低、成本高的问题,用一个通用的芯片将原来分散的芯片(CPU+MPU+xPUs)能力聚合到一起,将电子控制单元进行整合,是汽车电子设计的发展趋势。随着嵌入式系统需求的增加和复杂度的提升,其混合部署能力以及功能安全性也面临着新的挑战。一方面,传统的嵌入式系统通常使用单一类型的操作系统,难以满足复杂应用场景下的多功能需求。其中,混合部署指的是在一块 SOC 上运行多种异构执行环境的方式。另一方面,运行多种异构执行环境的架构增加了系统的复杂度,这导致了系统整体安全性的降低,嵌入式系统在运行过程中可能会受到各种攻击,例如恶意代码注入、网络攻击等,这些攻击可能会导致系统崩溃、数据泄露等严重后果。因此,如何设计一种能够兼顾混合部署能力和功能安全性的嵌入式系统架构,成为当前研究的热点问题。

本文将 DHR 结构与混合部署相融合,提供一种集灵活性和安全性为一体的 DHR-OS 架构,主要工作如下:(1)提出了一种主从操作系统混合部署框架,解决了嵌入式系统动态部署操作系统以及操作系统之间的协调工作问题;(2)设计了一套调度、分发、裁决为一体的关键任务安全执行机制,为 RTOS 侧任务的安全执行提供了条件;(3)基于国产飞腾 D2000 处理器实现了 DHR-OS 原型系统及其测试平台,通过测试证明了 DHR-OS 架构具备较高的可靠性。

2 相关工作

针对混合部署需求,目前从硬件架构上来看,多核可以分为同构多核和异构多核。同构多核指的是所有的 CPU 内核具有相同的架构,异构多核指的是 CPU 内核具有不同的架构。从软件设计来看,多核可以分为对称多处理(Symmetric Multiprocessing,SMP)和非对称多处理(Asymmetric Multiprocessing,AMP)。SMP 指的是所有的 CPU 内核运行同一套程序,AMP 指的是不同的 CPU 内核运行不同的程序[4]。多样的异构设计为混合部署提供了运行底座,在这种背景下,在嵌入式硬件平台上执行多个运行环境的混合部署模式应运而生。目前主要的混合部署模式实现方式有虚拟化模式[5]、异构多核模式[6]、同构多核模式[7]3 种。

虚拟化技术通过软件模拟硬件资源,使一台物理计算机能同时运行多个虚拟机,同时不同逻辑计算机可以运行不同的操作系统,提升了资源利用率、隔离性和管理灵活性。由于引入了虚拟机,这种结构可以实现硬件资源的复用,但是也增加了任务调用开销,影响了系统的实时性和性能[8]。

异构多核(Heterogeneous)混合部署模式指的是在异构硬件上运行非对称系统。异构多核处理器通常采用通用处理器+协处理器的结构。著名的异构多核处理器有 IBM 的 Cell、Xilinx 的 ZYNQ-7000 和 TI 的 OMAP 等[9]。在硬件层面上,异构多核处理器各个核心的用途已经确定,如移动终端采用的 AP+BP+CP 模式。其中,AP(Application Process)为手机中的应用处理器,操作系统和应用程序的任务都在 AP 上执行;BP(Baseband Process)为基带处理器,运行手机射频通讯控制软件,负责手机的无线通信功能,管理与网络的连接;CP(Co-Processor)为协处理器,运行 RTOS,负责处理实时任务[10]。因此,采用异构多核处理器的嵌入式系统均为非对称结构。

同构多核混合部署模式指的是在同构硬件上运行非对称系统,通过将不同特性的操作系统部署到架构相同的 CPU 核心上,为上层应用提供通用计算和实时控制的能力;在硬件层面实现了操作系统与硬件的完全解耦,即操作系统不需要绑定到特定的硬件核心,而是可以根据场景需求动态部署。

虚拟化模式的优点在于能够有效利用硬件资源,实现资源的灵活分配和隔离[11],但其缺点是增加了任务调用的开销,可能影响系统的实时性和整体性能,尤其在对实时性要求高的应用场景中显得不足。异构多核模式能够充分发挥不同核心的特性,适合特定任务的优化,但其固定性限制了系统的灵活性,各核心的用途已被预定义,可能导致资源利用不均衡,难以应对多变的应用需求。相比之下,同构多核模式通过在相同架构的 CPU 核心上部署不同特性的操作系统,实现了操作系统与硬件的解耦,提供了更好的灵活性和更高的资源利用率。这种模式能够根据具体场景动态调整部署,适应多样化的应用需求,特别是在需要实时控制和通用计算能力的场景中表现优越。因此,本文设计了一种基于同构多核的混合部署模式,即在不同核心上动态部署通用操作系统(General-Purpose Operating System,GPOS)和实时操作系统(Real-Time Operating System,RTOS)两种操作系统,并针对两种操作系统间的系统工作设计了驱动共享、中断管理、核间 RPC 调用等协同机制。

出于对操作系统安全性的考虑,在设备异构性和应用场景多样性不断增加的背景下,安全问题愈发突出。现有系统安全策略主要集中在安全系统框架构建、安全内核设计、可行执行隔离环境 3 个方面[12]。

安全框架的构建是指在设计初期就建立一个全面的安全策略框架,涵盖操作系统的各个层面。有研究人员提出在设备中增加安全模块,从硬件层面为用户提供安全侦测、分析、处理等安全功能,使整个系统拥有安全和可信的能力[13]。还有研究人员根据现阶段嵌入式设备存在的安全问题,提出了可信嵌入式设备设计的建议[14]。

安全内核的目标为增强操作系统内核的安全性,以抵御各种攻击。目前,相关的研究主要集中在轻量级安全内核的研究方面,其工作主要可分为两个方面。一方面是重新设计内核结构,增加内核本身的安全性。例如,有研究人员设计了安全内核原型系统[15],其可以提供身份认证、访问权限管理等多种安全功能,可用于多种嵌入式操作系统。另一方面,研究人员致力于在原有的内核结构上增加新的模块,以对内核

行为进行监测和处理。例如，有研究人员设计了轻量级的可信执行环境，用于保护原有内核的关键操作[16]。还有研究人员增加了额外的验证模块用于监测内核的运行时安全，从而保证内核关键操作与通信的正确运行[17]。

可信执行环境的目标为创建隔离的执行环境，以保护应用程序和数据的安全。其主要通过硬件和软件两种方式来实现。对于硬件隔离，有研究人员提出利用安全协处理器来确保关键工作的独立可信执行[18]。然而，使用安全协处理器会额外增加系统的功耗，并且主从处理器的通信延迟也会影响关键任务的实时性。对于软件隔离，研究人员主要通过软件错误隔离和硬件虚拟化来实现。软件错误隔离主要是在原有的软件架构上增加对控制流合法性的校验，并且对访存操作进行监测和控制[19]，从而实现上层应用之间数据流和控制流的隔离。

动态异构冗余（Dynamic Heterogeneous Redundancy，DHR）是我国邬江兴院士团队提出的一种新型安全体系架构[20]。该架构基于策略裁决的闭环迭代式动态重构机制结构，将因自身故障引起和因外界攻击造成的系统错误统一定义为“广义扰动”[21]。本文提出的动态异构冗余操作系统架构 DHR-OS 能够将关键任务分发到不同的 RTOS 核心上，同时由 GPOS 的裁决器处理不同 RTOS 的裁决结果。与传统安全策略的被动防御相比，DHR-OS 通过主动检测机制提供了一种新的系统架构解决方案，有效提升了系统的安全性和响应能力。

3 基于动态性的操作系统部署框架

传统的嵌入式运行平台通常只运行单一操作系统。但随着嵌入式系统向云协同和通用化方向发展，这种传统架构逐渐难以满足实时性和通用性的需求。为了解决这一问题，本文在 CPU 的同构核心上部署了不同功能的操作系统内核，并利用 OpenAMP 框架构建了高效的通信桥梁，实现了多个操作系统的协同运行。这种新型的系统设计允许每个操作系统专注于不同的任务和功能，通过 OpenAMP 的通信机制，它们能够实时、高效地共享信息和协同工作。这样的嵌入式系统架构不仅在满足实时性和功能性需求上具有显著优势，还为嵌入式系统的发展提供了更灵活、可扩展的解决方案。图 1 为 OpenAMP 通信框架图，该架构采用 Linux 的 remoteproc 子系统对远端核心实行生命周期管理，rpmsg 子系统为操作系统间建立通信桥梁。

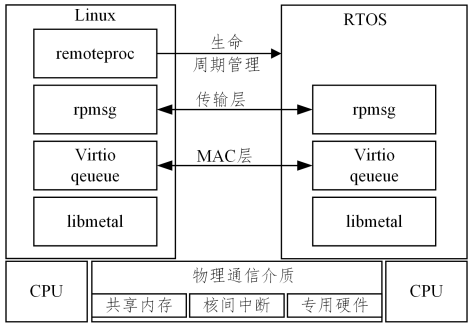

图 1 OpenAMP 通信框架图

Fig. 1 OpenAMP communication framework diagram

相比于传统的预先固定 RTOS 核心的部署方案，本文采用了一种更加灵活的动态配置机制，利用 ARM 架构的 SMC-CC（Secure Monitor Call Calling Convention）特性将动态管理的接口暴露给用户空间，从而实现了灵活的混合部署模式。这不仅能够更好地适应不断变化的应用场景，也为未来智能嵌入式应用提供了更强大的支持。

在建立了多 OS 混合部署框架之后，为了使得异构 OS 之间能够更好地进行协同工作，还需要设计一些基于 OpenAMP 的软件协同机制。其具体包括驱动共享机制、中断管理机制和核间 RPC 调用机制。

驱动共享机制通过 OpenAMP 实现跨 OS 的驱动资源共享，使得各个操作系统可以访问和使用共享的外设驱动。

中断管理机制建立跨 OS 的中断处理协调机制，确保中断事件能够被及时地传递和处理。

核间 RPC 调用机制提供高效的跨 OS 远程过程调用机制，使得不同操作系统之间能够进行直接的任务调用。

这些基于 OpenAMP 的软件协同设计，将进一步增强异构操作系统之间的协作能力，提高整个混合部署系统的可靠性和灵活性。

3.1 驱动共享机制

在嵌入式系统中，多操作系统协同运行已经成为一种常见的架构。通过利用不同操作系统的优势，这种架构能够显著提升系统的整体性能和功能。然而，实现多操作系统协同运行的关键在于设计高效的驱动共享机制。

通常采用自旋锁的形式，需要手动申请和释放锁。这种方式简单易懂，但存在一些缺陷。首先，自旋锁会消耗 CPU 资源，尤其是在高负载情况下，性能损耗较为明显。其次，当多个操作系统同时竞争同一资源时，自旋锁的竞争会加剧，导致系统效率下降。为了弥补传统驱动共享机制的不足，我们利用 OpenAMP 设计了一种带优先级的时分复用锁，并引入可配置份额的时间片。带优先级的锁可以根据不同操作系统的优先级分配锁的使用权，确保高优先级操作系统能够优先访问资源。这样不仅可以提高系统的安全性，还能确保关键任务的及时响应。可配置份额的时间片锁则可以根据不同操作系统的需求，灵活调整时间片分配比例。例如，可以将时间片分配为 100∶0，相当于将锁完全分配给高优先级操作系统，退化为 FreeRTOS 优先锁。这种方式不仅能够有效减少锁的竞争，提高系统性能，还可以根据实际需求灵活调整资源分配策略，增强系统的灵活性。具体的设计框架图如图 2 所示。

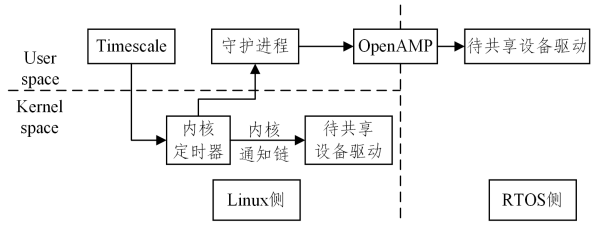

图 2 驱动共享机制框架图

Fig. 2 Driver sharing mechanism framework diagram

总的来说，改进方案能够有效弥补传统驱动共享机制的

不足。通过设计带优先级的锁和可配置份额的时间片的优先级锁，该方案不仅能够提高系统的性能和安全性，还可以增强系统的灵活性。这不仅为多操作系统协同运行提供了更好的支持，还能充分发挥多操作系统协同的优势，提升嵌入式系统的整体性能。

3.2 中断管理机制

在多操作系统协同运行的嵌入式系统中，中断管理是一个非常重要的环节。不同操作系统对中断的处理机制和优先级管理可能存在差异，如果处理不当，会导致系统行为异常。由于需要优先保障实时性，因此在 RTOS 侧添加一个虚拟中断管理层 VGIC(Virtual General Interrupt Controller)，用于屏蔽不同操作系统对中断处理机制的差异，并将中断信号准确路由至目标核心。VGIC 的设计思路如下：VGIC 维护一个虚拟中断号到实际中断号的映射表，FreeRTOS 接收到中断请求时，首先通过映射表找到对应的实际中断号，然后再进行处理。这样可以屏蔽 FreeRTOS 和 Linux 之间中断号的差异。中断优先级管理：VGIC 还需要管理不同操作系统之间的中断优先级，通过设置虚拟中断的优先级，可以确保高优先级的中断优先得到响应，提高系统的实时性和稳定性。中断注册和注销：VGIC 应该提供中断注册和注销的接口，使得 FreeRTOS 和 Linux 可以动态注册和注销中断处理程序，增强系统的可配置性和扩展性。对于 RTOS 侧的中断，直接调用自己编写的 API，通过设置 GIC 中断控制器来注册和注销中断。对于 Linux 侧的中断，系统通过 OpenAMP 建立的通信通道来通知 Linux 侧进行相应的处理。通过以上设计，VGIC 可以有效解决 RTOS 和 Linux 之间中断管理的差异，提高系统的实时性、稳定性和资源利用率。同时，VGIC 的可配置性也使得中断管理机制更加灵活，可以根据不同的应用场景进行定制和优化。总的来说，在 FreeRTOS 中添加 VGIC 虚拟中断管理层，不仅可以解决多操作系统协同运行中的中断管理问题，还可以提升系统的整体性能和可扩展性。具体的设计框图如图 3 所示。

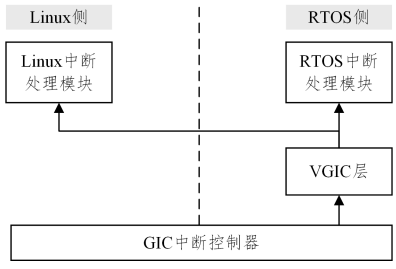

图 3 中断管理机制框架图

Fig. 3 Interrupt management mechanism framework diagram

3.3 核间任务 RPC 调用设计

在多操作系统协同运行的嵌入式系统中，不同操作系统之间的任务交互是一个重要的设计考量。为了实现跨操作系统的任务调用，可以利用基于 OpenAMP 的跨核间通信机制，设计一套可靠高效的 RPC(Remote Procedure Call)调用方案。

具体的 RPC 调用方案如下：为了实现 Linux 调用 RTOS 侧的函数，在 Linux 侧设置一个同名函数。在 Linux 侧的同名函数内部，首先将调用参数进行序列化，然后通过 OpenAMP 将参数传递给 RTOS 侧。RTOS 侧接收到调用请求后，根据参数执行相应的函数，并将返回值进行序列化后通过 OpenAMP 回传给 Linux 侧。Linux 侧再对返回值进行反序列化，完成一个跨操作系统的远程函数调用。对于带有指针的参数，需要先将指针指向的数据类型进行序列化，然后传递给 RTOS 侧，RTOS 侧再对接收到的序列化数据进行反序列化处理。这样既可以保证参数的完整性，又可以实现跨操作系统的指针参数传递。通过这种基于 OpenAMP 的 RPC 调用机制，可以实现 Linux 和 RTOS 之间的高效、可靠的任务交互。该方案不仅屏蔽了底层通信细节，提高了开发效率，同时也确保了跨操作系统调用的实时性和稳定性，为嵌入式系统的协同运行提供有力支撑。具体的设计图如图 4 所示。

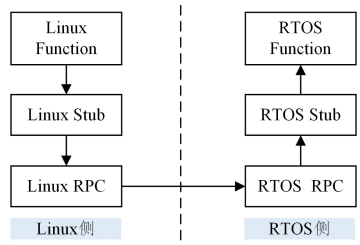

图 4 核间任务 RPC 调用框架图

Fig. 4 Inter-core task RPC call framework diagram

4 基于异构冗余性的任务安全执行环境

4.1 多冗余度的 DHR-OS 设计

DHR 是一种典型的容错架构，它由功能等价的异构执行单元、策略分发、策略裁决和策略调度等模块组成。DHR 通过“异构”和“冗余”的执行，可以确保在绝大部分攻击场景下，只影响个别执行单元的运行状态或输出结果。这种对个别执行单元的影响也能通过裁决器模块被识别为异常执行个体并屏蔽其输出，异常执行个体也会被策略调度模块恢复为正常执行单元，或者从异构执行单元池中删除。同时，动态调度策略减少了多个执行单元因共性缺陷或协同攻击而产生相同错误输出的可能性，从而降低了短期攻击成功的概率。实际上，DHR 防御策略将已知或未知的攻击事件统一归纳为系统故障事件，这种处理方式不再针对某些特定的攻击手段做被动防御，而是主动分析冗余执行单元对某一任务的执行结果，为系统安全问题提出了一种新的解决思路。在工程实现时，异构执行体通常会采用三模冗余(TMR)的方式设置。不过，TMR 结构也存在一些问题，比如当有一个执行单元出现故障需要从异构执行单元池中删除时，剩下的两个执行单元再次出现输出不一致时，从两个剩余的执行结果随机选择一个，这种随机输出错误率高达 50%，难以接受。因此，本文设计了一种基于多模冗余(Multiple Modular Redundancy，MMR)的 DHR-OS 结构，其在 TMR 结构的基础上，将原来的 3 个冗余 OS 执行体按照 Linux 侧任务吞吐量的大小增加到 3～6 个异构 OS 执行体，可有效缓解上述问题。

DHR 的典型结构如图 5 所示。其中，策略调度模块决定异构执行体的数量和类别。执行体确定后，系统根据分发策略将某一个或某一种任务分发到不同的异构执行体上，最终

策略裁决模块从这些执行结果中推选出最终的执行结果，并且将裁决信息反馈到策略调度模块中。例如某个异构执行体异常，策略调度模块会将这个异构执行体恢复，或从异构执行单元池中删除。

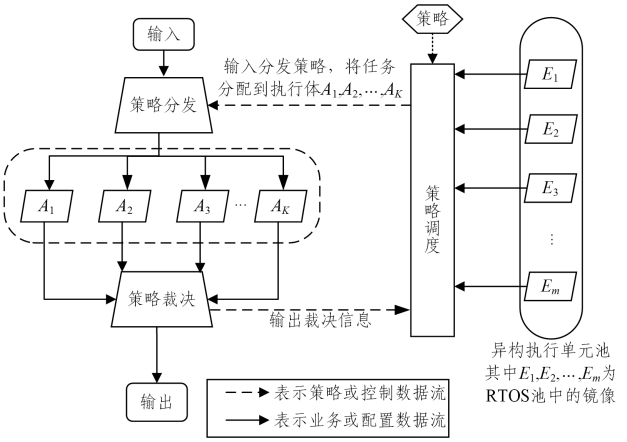

图 5 DHR 典型结构

Fig. 5 Typical structure of DHR

本文设计的 DHR-OS 架构如图 6 所示，需要说明，本文主要侧重于关键任务在 RTOS 核心的安全执行。架构中的策略调度模块、策略分发模块以及基于加权投票的共识算法所代表的裁决器模块均在 GPOS 侧实现，GPOS 采用 Linux 操作系统。在 DHR-OS 架构中，Linux 侧的功能涵盖了策略分发、策略裁决以及策略调度等关键模块。作为 DHR-OS 系统与外部唯一的数据传输接口，Linux 侧上述模块的安全性是设计的首要原则。攻击者一旦绕过 DHR 架构的防御，直接对上述模块发起攻击，便可获取 DHR-OS 的完全控制权，因此必须确保其安全性。此外，相较于常规的操作系统在满足 OS 控制场景的实时性和多样性要求的同时，调度裁决器还应具备一定的灵活性，以确保在完成输入分发、输出裁决和策略调度、同步处理等功能时，不会影响整体的运行效率。针对上述设计目标，本文拟采用在 Linux 侧部署调度裁决器模块的方案。具体来说，调度裁决器将承担以下关键职责：策略分发负责将来自外部的策略请求分发到相应的异构 OS 镜像进行处理；策略裁决负责收集并评估异构 OS 镜像的处理结果，并根据特定的共识算法做出最终的裁决；策略调度根据裁决结果，将策略指令下发到相应的异构 OS 镜像，协调其执行。

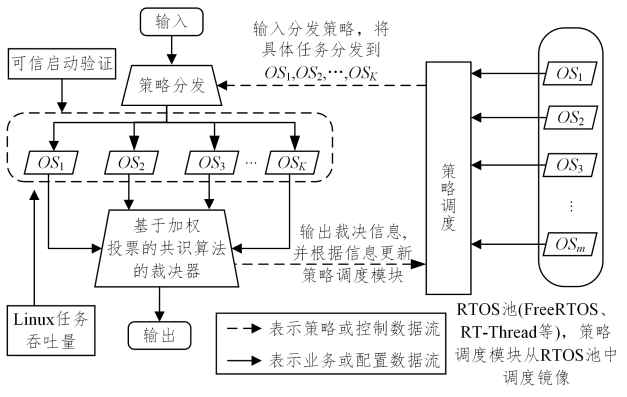

图 6 DHR-OS 架构图

Fig. 6 DHR-OS architecture diagram

通过上述设计，裁决器模块能够充分利用 Linux 操作系统的安全机制，在保证安全性的同时，也能够满足 DHR-OS 架构对实时性和灵活性的要求，为整个系统提供可靠的核心控制服务。RTOS 内核池由 SylixOS，FreeRTOS 和 RT-Thread 若干个 RTOS 组成。以一个 8 核心 CPU 为例，通常 0 号 CPU 固定运行 Linux；7 号 CPU 为一个常驻的 FreeRTOS 镜像，负责系统的实时任务；1－6 号 CPU 在调度器无分发请求时运行 Linux 操作系统，当调度器有分发请求时，根据 Linux 侧任务的吞吐量随机从内核池中选择 3～6 个冗余内核执行相同任务，由 Linux 侧的策略分发模块将具体任务分发到各个 RTOS 执行体上，并根据加权投票的共识算法裁决出一个最终结果。同时，裁决器模块对得出错误执行结果的内核进行标记并反馈给策略调度模块，将异常的 RTOS 核心进行恢复或将其移出异构内核池，保障整个系统的稳定和安全。为了防止 RTOS 镜像启动时被篡改，在每次启动 RTOS 镜像前，Linux 侧还会基于 SM4 算法对 RTOS 镜像进行可信启动校验，相当于 Linux 策略调度模块以软件可信根的形式保证了 RTOS 镜像的可信启动。同时，利用 OpenAMP 模块实现对异构 RTOS 镜像的管理。OpenAMP 主要通过 read/write 和 ioctl 系统调用 Linux 生成的 sysfs 文件进行操作，因此可以利用 Linux 的 SELinux 机制来保证调度器的安全性。

4.2 策略调度模块设计

策略调度模块主要负责将来自外部的策略请求分发到相应的异构 OS 镜像进行处理。具体设计如下：0 号核心常驻 Linux 操作系统；7 号核心常驻 FreeRTOS 操作系统；策略调度模块将在其余 6 个核心中部署 RTOS 镜像执行任务。

策略调度模块根据 Linux 侧的系统吞吐量来决定冗余执行单元的数量，具体地，通过 /proc/loadavg 文件中的负载平均值来确定执行单元的数量。首先，从 /proc/loadavg 中读取最近 1 min 的平均负载值(avg1)，系统有 8 个核心，则可用调度核心数为 max(3,8-avg1)。例如，当 avg1 为 4.5 时，系统可用调度核心数为 3，这意味着可以调度 3 个核心来执行 RTOS 镜像任务。此机制确保了系统在不同负载条件下的调度效率和资源利用率。此外，具体采用的 RTOS 的种类及其应绑定的硬件 CPU 号也将通过随机时间种子进行确定，以实现动态和灵活的调度策略。

具体地，定义一个包含可用 RTOS 种类的列表，如["FreeRTOS","SylixOS","RT-Thread"]，以及硬件 CPU 核心的总数量。然后，将当前时间戳作为随机种子初始化随机数生成器，以确保每次运行的随机性。接着，从 RTOS 列表中随机选择一个 RTOS 种类，并从可用的 CPU 核心中随机选择一个 CPU 号进行绑定。最后，输出选定的 RTOS 种类及其对应的 CPU 号。该算法通过随机选择实现了灵活的调度策略，能够根据系统需求动态调整使用的 RTOS 和绑定的 CPU 核心。

4.3 基于加权投票的共识算法的裁决器设计

在多操作系统协同运行的裁决器设计中，需要采用一种高效可靠的共识算法来维护系统的一致性。本节提出了一种基于加权投票的共识算法，以解决镜像可信度随时间降低和历史沦陷次数增加的问题。加权投票共识算法的设计思路如下。

镜像可信度 δ：每个镜像都有一个初始可信度 δ_0，该值会随着镜像运行时间的增加而逐渐降低。可信度值的计算式为：

$$\delta=\delta_0 * a \tag{1}$$

其中，时间衰减系数 a 是一个小于 1 的常数，用于表示镜像可信度随时间降低的趋势。

沦陷系数 k：每个镜像都有一个沦陷系数，该值会随着镜像历史沦陷次数的增加而降低。沦陷系数的计算式为：

$$k=(1\rightarrow 0.01) * n \tag{2}$$

其中，n 为历史沦陷次数。也就是说，每次镜像沦陷，沦陷系数就会减少 0.01。

加权投票机制：在进行共识时，每个镜像的投票权重 V 由其可信度和沦陷系数共同决定。权重的计算式为：

$$V=\delta * k \tag{3}$$

权重越高的镜像，其投票的影响力也就越大。

多数投票共识：在进行共识时，系统会收集所有镜像的投票结果，根据加权投票机制计算出每个选项的总得分。最终，得分最高的选项即为共识结果。

通过上述设计，该共识算法能够动态调整每个镜像的投票权重，以应对镜像可信度随时间降低和历史沦陷次数增加的问题。

可信度较高且历史记录良好的镜像，其投票权重较大，对共识结果有较大的影响力。可信度较低或历史记录不佳的镜像，其投票权重较小，对共识结果的影响较小。

这种加权投票机制可以有效提高共识算法的可靠性和鲁棒性，为裁决器设计提供稳定的一致性保证。

5 DHR-OS 原型系统及测试

5.1 实验环境

本文实验环境的硬件为：CPU 腾锐 D2000 集成 8 个 FTC663 内核，兼容 64 位 ARMv8 指令集，主频 2.3-2.6GHz. 8GB RAM；SOC 内部内置 GIV-V3 中断控制器，8 块核心通过总线访问同一块内存。软件环境为：Linux5.10，Freertos 8.0.0。本文分别从部署框架和关键任务安全两方面来分别进行测试。

5.2 功能测试

功能测试主要分为两大类：操作系统混合部署功能测试和 DHR-OS 功能测试。在操作系统混合部署功能测试中，重点测试了 Linux 与 RTOS 之间的协同机制，如表 1 所列。

表 1 协同功能测试

Table 1 Cooperative function testing

	测试项
协同功能测试	RTOS 镜像调用测试
	Linux 和 RTOS 间通信测试
	虚拟中断转发测试
	驱动共享示例测试
	RPC 远程函数调用测试

在调度机制测试中，采用了“白盒插桩”的方法来模拟可能出现的异常情况，以触发系统的防御机制。白盒插桩的错误输出并非由实际扰动引起，而是通过软件配置人为产生。采用这种方式，一方面降低了测试过程中产生错误结果的复杂性，另一方面更有效地模拟了未知攻击或故障造成的意外扰动，更符合 DHR-OS 架构的防御目标。功能测试结果如表 2 所列，测试结果表明，操作系统的各项基本功能均能通过测试用例验证，DHR-OS 在异常发生时能够顺利恢复到正常工作状态。

表 2 DHR-OS 机理测试

Table 2 DHR-OS Mechanism Testing

	测试项
DHR-OS 机理测试	策略调度模块功能测试
	策略分发模块机制测试
	调度裁决模块机制测试

5.3 抗攻击测试

对于 DHR-OS，外部攻击所造成的影响将直接反映在各个 RTOS 输出的数据中。当单个 OS 被攻击处理异常状态时，称为差模攻击；当多个 OS 被攻击时，称为多模攻击；当所有 OS 被攻击输出都是异常时，称为共模攻击。本文通过模拟内部故障或外部攻击，分别对系统进行差模攻击、多模攻击、对模攻击，主要的实验设计思路如下。

主操作系统为 Linux，动态调度 3～6 个 RTOS 核心。通过配置中断源，利用 Linux 发送的 IPI 信号触发软件中断，向特定的 RTOS 核心发送中断信号，以模拟恶意代码的注入。同时，在每个 RTOS 核心内插入随机执行宕机指令，模拟内部故障。在外部中断和内部故障同时发生的情况下，执行一组预定义的任务，记录系统的输出结果。实验结果如表 3 所列，包括外部中断情况、内部故障、系统输出结果及成功与否的状态。

表 3 DHR-OS 抗攻击测试

Table 3 DHR-OS attack resistance testing

实验编号	调度核心组合	外部中断触发组合	内部故障指令	预期结果	系统结果	是否成功
1	Core 1,2,3	Core 1	无	1	1	是
2	Core 1,2,3	无	Core2	1	1	是
3	Core 1,2,3,4	Core1,2	无	1	1	是
4	Core 1,2,3,4	Core1	Core2	1	1	是
5	Core 1,2,3,4,5	Core1,2	无	1	1	是
6	Core 1,2,3,4,5	Core1	Core2	1	1	是
7	Core 1,2,3,4,5	Core1,2,3	无	1	1	是
8	Core 1,2,3,4,5,6	Core1,2	Core3	1	1	是
9	Core 1,2,3,4,5,6	Core1	Core2,3	1	1	是
10	Core 1,2,3,4,5,6	Core 1,2,3,4	无	1	2	否
11	Core 1,2,3,4,5,6	无	Core 1,2,3,4	1	裁决器等待超时	否
12	Core 1,2,3,4,5,6	Core 1,2,3,4,5,6	无	1	2	否
13	Core 1,2,3,4,5,6	无	Core 1,2,3,4,5,6	1	裁决器等待超时	否

实验结果表明，无论在 1－2 号实验的差模攻击场景还是 3－9 号实验的多模攻击场景下，DHR-OS 都能表现出很好的抗攻击特性；在 10 号与 12 号实验场景中，超过半数的 RTOS 核心同时被攻击时，由于冗余度不足，加权投票算法无法有效区分正常与异常结果，导致裁决器输出错误结果 2；在 11 号与 13 号场景中，超过半数的 RTOS 系统出现内部故障，无法正确输出结果供裁决器使用，裁决

器在等待超时后终止本次裁决操作。

需要说明的是，本文使用的白盒插桩测试方法将攻击场景理想化，即直接反映到各 RTOS。相比之下，实际攻击要比本实验中使用的理想化攻击场景更加困难。因此，本文所提的 DHR-OS 架构在实际的抗攻击性方面将会有更大的提升。

结束语 在当今快速发展的信息时代，嵌入式系统作为各种智能设备和网络物理系统的核心，其安全性和灵活性成为了研究与开发的焦点。本文提出的 DHR-OS 架构，通过动态异构冗余的设计，成功地在多核 CPU 上实现了异构操作系统的混合部署，并有效整合了 OpenAMP 框架，以促进系统间的无缝通信。这一创新性架构不仅满足了嵌入式系统对实时性与功能多样性的需求，同时也极大地提升了系统在面对安全威胁时的抵御能力。通过设计多模裁决器与分布式共识算法，DHR-OS 架构在保障系统正常运行的同时，能够有效识别并应对各类潜在攻击，提高了系统的安全性和鲁棒性。原型系统的实现与测试结果进一步验证了该架构在实际应用中的可行性。本研究已实现较高的任务安全执行与抗攻击能力，但在实际应用中仍需进一步增强系统的安全性与防御能力。未来将结合机器学习与大数据分析技术，对差模、多模及共模攻击进行异常行为建模与实时检测，从而提升系统对未知攻击与故障的自适应防御能力。

参 考 文 献

[1] PINTO R, TORRES P M B, LOHWEG V. Closing editorial: Advances and future directions in autonomous systems for cyber-physical systems and smart industry[J]. Applied Sciences, 2024, 14(22): 10673.

[2] SINGH J, SINGH A, SINGH H, et al. Implementation and evaluation of a smart machine monitoring system under industry 4.0 concept[J]. Journal of Industrial Information Integration, 2025, 43: 100746.

[3] CHANDRA S, SAMUEL M, EUGENE V E, et al. Review of the security of backward-compatible automotive inter-ecu communication[J]. IEEE Access, 2021, 9: 114854-114869.

[4] QU W, YU F H. Survey of Research on Asymmetric Embedded System Based on Multi-core Processor[J]. Computer Science, 2021, 48(S1): 538-542.

[5] WANG Z. Application of Virtualization Technology in Computer Systems[J]. Integrated Circuit Applications, 2024, 41(10): 66-67.

[6] ROHLIN A, FAHLGREN H, PERICÀS M. High performance scheduling of mixed-modedags on heterogeneous multicores[J]. arXiv: 1901.05907, 2019.

[7] ZHAO W F, ZHAO Y. Research on key technologies of multi-core processor systems[J]. Digital Technology and Applications, 2023, 41(3): 123-125.

[8] ZHANG M Y, ZHANG Q Y, MENG Z Q. A survey of research on real-time dual-OS architecture for embedded platform[J]. Acta Electronica Sinica, 2018, 46(11): 2787-96.

[9] KONG X Z. Research and Implementation of Embedded Real-time Operating System for SMP Architecture DSP[D]. Xi'an: Xidian University, 2013.

[10] WU J X. Introduction to Cyberspace Mimic Defense[M]. Beijing: Science Press, 2017.

[11] LI S Y. Application and development trends of virtualization technology in computer network security[N]. Anhui Science and Technology News, 2024-10-18.

[12] PENG A N, ZHOU W, JIAY, et al. Survey of the Internet of things operating system security[J]. Journal on Communications, 2018, 39(3): 22-34.

[13] BABAR S, STANGO A, PRASAD N, et al. Proposed embedded security framework for Internet of things(IoT)[C]// 2011 2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronics Systems Technology(Wireless VITAE). 2011: 1-5.

[14] JIN Y. Embedded system security in smart consumer electronics [C]// The 4th International Workshop on Trustworthy Embedded Devices. 2014.

[15] LIU S. Design and development of a security kernel in an embedded system[J]. International Journal of Control & Automation, 2014, 7(11): 49-58.

[16] AZAB A M, SWIDOWSKI K, BHUTKAR R, et al. SKEE: a lightweight secure kernel-level execution environment for ARM [C]// NDSS. 2016.

[17] BATES A, TIAN D, BUTLER K R B, et al. Trustworthy whole-system provenance for the Linux kernel[C]// Usenix Conference on Security Symposium. 2015: 319-334.

[18] DYER J G, LINDEMANN M, PEREZ R, et al. Building the IBM 4758 secure coprocessor[J]. Computer, 2001, 34(10): 57-66.

[19] ZHAO L, LI G, SUTTER B D, et al. ARMor: fully verified software fault isolation[C]// The International Conference on Embedded Software. 2011: 289-298.

[20] WU J X. Introduction to Cyberspace Mimic Defense[M]. Beijing: Science Press, 2017.

[21] OUYANG L, SONG K, LAN J L. Design and Implementation of Microcontroller Based on Dynamic Heterogeneous Redundancy Architecture[J]. Acta Electronica Sinica, 2023, 20(9): 144-159.

HE Ruiqi, born in 1999, postgraduate, is a member of CCF (No. Q0031G). His main research interests include embedded system design and so on.

ZHANG Kailong, born in 1977, professor, is a distinguished member of CCF (No. 08116D). His main research interests include adaptive embedded real-time computing and intelligent systems.

（责任编辑：柯颖）