

基于FreeRTOS优化的嵌入式实时操作系统设计

魏杰¹, 尹力², 曹晋豪¹, 李凤岐¹, 王德广¹

(1. 大连交通大学 轨道智能工程学院; 2. 大连交通大学 机械工程学院, 辽宁 大连 116028)

摘要: 针对当前嵌入式操作系统实时性能与嵌入式系统需求不匹配的问题, 深入剖析了FreeRTOS嵌入式实操作系统内核的性能瓶颈, 提出了一系列内核性能优化创新策略。首先, 在FreeRTOS的基础上, 结合轮询调度机制与基于优先级的抢占式调度算法进行任务调度, 以提升系统实时性; 其次, 引入内存两级分割策略算法(TLSF)算法与内存池机制, 动态分配与回收管理内存, 在减少内存碎片的同时快速分配、回收内存; 最后, 增加任务直接通知通信机制, 增强系统的实时响应能力。基于上述创新策略, 采用分层设计实现了基础且高效内核功能的基于Cortex-M架构的嵌入式操作系统CMRTOS。硬件仿真与实例表明, CMRTOS相较于FreeRTOS, 在内存资源利用效率、实时性和稳定性方面提升显著。

关键词: 嵌入式操作系统; 内核功能; 任务调度; 内存分配; 通信机制; 实时性

DOI: 10.11907/rjtk.242054

扫描二维码阅读全文:



中图分类号: TP316.2

文献标识码: A

文章编号: 1672-7800(2026)003-0107-08

Design of Embedded Real-Time Operating System Based on FreeRTOS Optimization

WEI Jie¹, YIN Li², CAO Jinhao¹, LI Fengqi¹, WANG Deguang¹

(1. School of Rail Intelligent Engineering, Dalian Jiaotong University;

2. School of Mechanical Engineering, Dalian Jiaotong University, Dalian 116028, China)

Abstract: Aiming at the problem that the real-time performance of current embedded operating systems does not match the requirements of embedded systems, the performance bottleneck of the FreeRTOS embedded real operating system kernel is deeply analyzed, and a series of innovative strategies for kernel performance optimization are proposed accordingly. Based on FreeRTOS, a polling scheduling mechanism and a priority-based preemptive scheduling algorithm are combined to schedule tasks to improve the real-time performance of the system; a two-level segmented free list (TLSF) algorithm and a memory pool mechanism are introduced to dynamically allocate and recycle memory, reducing memory fragmentation while achieving fast memory allocation and recycling; a task direct notification communication mechanism is added to enhance the real-time response capability of the system. Based on the above innovative strategies, a Cortex-M-based embedded operating system CMRTOS is designed. The system architecture adopts a layered design to realize basic and efficient kernel functions. Hardware simulation and example verification show that compared with FreeRTOS, CMRTOS shows significant improvements in memory resource utilization efficiency, real-time performance and stability.

Key Words: embedded operating system; kernel functionality; task scheduling; memory allocation; communication mechanism; real-time

收稿日期: 2025-01-16

基金项目: 辽宁省国际科技合作计划项目(2022JH2/10700012); 辽宁省应用基础研究计划项目(2023JH2/101300188); 辽宁省教育厅基础研究项目(JYTMS20230011); 辽宁省科研基金项目(LJKQZ20222447)

作者简介: 魏杰(1999-), 女, 大连交通大学轨道智能工程学院硕士研究生, 研究方向为智能信息系统、嵌入式操作系统; 尹力(1995-), 男, 大连交通大学机械工程学院博士研究生, 研究方向为工业物联网、智能信息系统; 曹晋豪(1999-), 男, 大连交通大学轨道智能工程学院硕士研究生, 研究方向为智能信息系统、人工智能; 李凤岐(1974-), 男, 博士, CCF专业会员, 大连交通大学轨道智能工程学院教授、博士生导师, 研究方向为智能信息系统、区块链; 王德广(1968-), 男, 大连交通大学轨道智能工程学院副教授, 研究方向为信息安全、区块链。本文通讯作者: 李凤岐。

0 引言

随着信息技术飞速发展,网络化与智能化趋势显著增强,传统电子技术正经历深刻变革,嵌入式系统凭借其独特优势,已广泛应用于智能手机、智能汽车、智能家居、工业制造等领域^[1-4]。FreeRTOS 作为嵌入式实时操作系统,因开源、轻量级和可扩展性受到青睐,但随着应用场景复杂化,在内核性能上暴露出调度效率低、内存管理差、实时响应不足等问题,限制了其在高实时性应用中的发展^[5,6]。为此,研发先进、高效的新型嵌入式操作系统,对推动嵌入式技术持续发展具有重要意义。

目前,研究人员提出了多种嵌入式操作系统设计方案,以提升系统性能。Karla 等^[7]将自定义调度算法集成到开源实时操作系统 FreeRTOS,通过自定义动态调度策略提升系统在欠载和过载情况下的性能,但因内存分配效率方面的局限性,系统实时响应能力未能得到实质性提升。Chunmao 等^[8]在 FreeRTOS 基础上设计了专为多核平台打造的操作系统,促进了多核处理器在嵌入式领域的应用,增强了系统安全性和适用范围,但实时性保障仍存在挑战。

为了解决上述问题,本文选用基于 Cortex-M 内核的高性能微控制器 STM32F407 作为核心硬件平台,通过对 FreeRTOS 内核进行优化,设计了一种基于 Cortex-M 架构的嵌入式操作系统 CMRTOS^[9,10]。

1 CMRTOS 系统架构

CMRTOS 是一个基于 FreeRTOS 的操作系统,架构设计与 FreeRTOS 相似,分为硬件抽象层、内核层和应用层 3 个层次^[11]。其中,硬件抽象层封装了对底层硬件平台的直接访问接口,实现了硬件无关性,使上层软件能以一种统一、抽象的方式与具体硬件交互,极大提升了系统可移植性和灵活性;应用层提供了丰富多样的应用程序与服务接口,直接面向用户或特定应用需求,实现了复杂的功能逻辑和交互体验;内核层集成了任务调度、内存管理、时间管理、通信管理等关键功能模块,以管理、控制系统的硬件和软件资源,确保系统高效、稳定地运行。

内核的功能模块设计结构来源于 FreeRTOS,但 CMRTOS 在此基础上引入了更多优化机制,整个 CMRTOS 操作系统的软件架构呈现出高度的模块化特征,各功能模块之间耦合度较低,便于独立开发和维护。具体地,系统采用标准化接口与协议策略,极大简化了新的传感器、控制器及外部设备接入的复杂性,不仅促进了系统对不同类型硬件设备的广泛兼容性,还拓宽了系统的潜在应用领域,赋予了系统更高的灵活性与可扩展性。

任务调度模块承担着管理系统中任务执行顺序的关

键职责,设计与实现方式对操作系统的实时性能影响很大。调度器用于管理任务状态,并在必要时执行上下文切换操作,实现高效、有序的任务调度策略,确保系统满足严格的实时性要求。

CMRTOS 采用了基于优先级的抢占式调度与轮询调度机制结合的方式,可有效、及时地处理高优先级任务,确保系统整体的实时性。同样,FreeRTOS 采用了优先级抢占式调度,但 CMRTOS 增加了轮询机制,确保在处理高优先级任务时不会忽略低优先级任务,从而优化了调度效率。

时间管理模块在系统时间跟踪和管理中发挥了关键作用。CMRTOS 的时间管理模块采用了与 FreeRTOS 相同的设计结构,结合了硬件定时器与软件定时器的双重优势,以应对资源受限环境下对时间精确性、任务调度效率的高要求^[12]。同时,系统使用 SysTick 定时器作为系统硬件定时器,利用硬件定时器的高精度特性直接接入系统时钟源,周期性地提供系统时钟节拍,确保系统时间的精确同步与更新,为系统提供一个稳定可靠的时间基准^[13]。软件定时器则作为硬件功能的补充与扩展,通过软件逻辑实现更灵活的时间管理与任务调度策略。软硬结合的设计策略不仅提升了系统的时间管理能力,还增强了任务执行的灵活性、及时性,为嵌入式系统在复杂多变的环境中高效运行提供有力保障。

内存管理模块的职责在于高效地组织内存空间,精确地追踪内存使用状况,实现快速可靠的内存分配与释放机制。内存管理策略设计必须兼顾效率与可预测性,确保内存操作能在极短时间内完成,不会因资源竞争或管理不当引入延迟。同时,为了避免标准内存管理方案(如标准库中的 *malloc()* 和 *free()* 函数)在实时环境中可能导致的内存碎片化问题,CMRTOS 采用内存两级分割策略算法(Two-Level Segregate Memory Allocator, TLSF)动态分配内存,有效优化了内存分配与释放效率^[14,15]。实验表明,TLSF 算法相较于 FreeRTOS 默认的标准内存分配机制 *heap_4* 而言,在保证实时性、减少碎片方面提升显著^[16]。

通信管理模块在任务间的同步、资源共享及事件通知等方面起着至关重要的作用。CMRTOS 系统中集成了多样化的任务间通信与同步机制,包括消息队列、信号量机制、事件组等高级同步原语,均是从 FreeRTOS 借鉴而来的基础组件^[17]。为了优化资源利用效率,创新性地提出了任务直接通知机制,使任务间的通信更高效,尤其在资源受限的嵌入式环境中能更好地满足实时性要求。CMRTOS 具体的系统架构如图 1 所示。

2 系统主要模块

2.1 内存管理模块

为了减少内存碎片,CMRTOS 使用内存池管理技术,

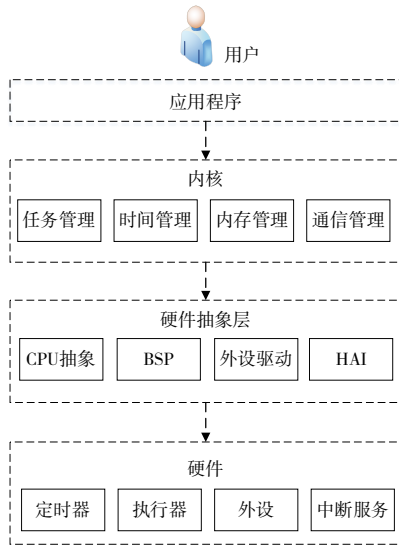


Fig. 1 CMRTOS system architecture
图1 CMRTOS系统架构

预先规划、分配一块连续的大容量内存区域作为内存池的基底。首先,该区域被精细地划分为多个固定大小的内存块,共同构成一个结构清晰、组织有序的内存池系统,以确保内存分配与释放操作能以块为单位执行,降低内存碎片产生;其次,将动态内存堆进一步细分为多个等级的内存池,每个等级的内存池专门负责管理特定大小范围的内存块,不仅增强了系统对各类内存请求的快速响应能力,还促进了内存资源的优化利用。不同等级的内存池如图2所示。

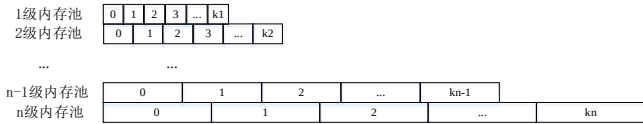


Fig. 2 Different levels of memory pools
图2 不同等级内存池

在内存池技术的基础上,系统采用TLSF算法动态分配内存,将动态内存分配与回收时间复杂度都降到 $O(1)$ 时间复杂度,既满足了系统实时性要求,又减少了内部碎片累积。首先,定义全局的`tlsf_control`结构来管理TLSF内存池,并将所有对TLSF的调用封装在`irq_disable()`与`irq_restore()`函数对中,确保在多线程环境下操作的原子性与线程安全性;其次,采用两级位图与链表管理内存池,根据需要的内存大小,利用两级分割算法计算第一级数组(First List, fl)和第二级数组(Second List, sl);最后,采用Good-fit策略寻找到大小合适的空闲块链。Good-fit是一种平衡策略,通过设置一个上限、下限来定义一个“好”的块,结合了Best-Fit与First-Fit的优点,在查找空闲内存块时既考虑了块的适配性,又提升了查找效率^[18]。

图3为TLSF算法的数据结构。第一级将内存大小按2的幂次方划分一个粗粒度的范围,第二级在第一级的基础上进行线性化的细粒度划分,由可配置的二级数组索引

(Second Level Index, SLI)参数确定划分个数,每个链表数组都有一个相关联的位图来记录内存块大小和位置,以及是否被使用。

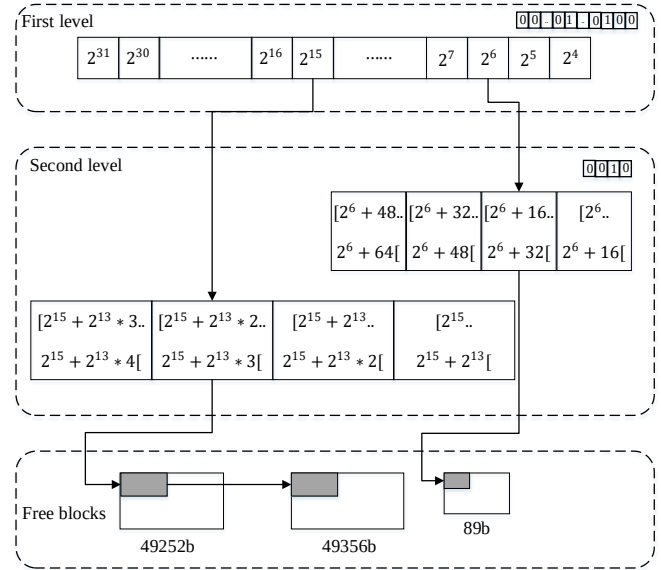


Fig. 3 TLSF algorithm data structure
图3 TLSF算法数据结构

本文在CMRTOS中集成TLSF算法优化内存管理,通过重载内存分配与释放函数`cmPortMalloc()`、`cmPortFree()`,分别调用TLSF提供的`tlsf_malloc()`、`tlsf_free()`内存管理接口。在内存释放过程中,`tlsf_free()`函数自动合并相邻的空闲内存块,减少内存碎片,提升内存利用率。内存管理的实现过程如图4所示。

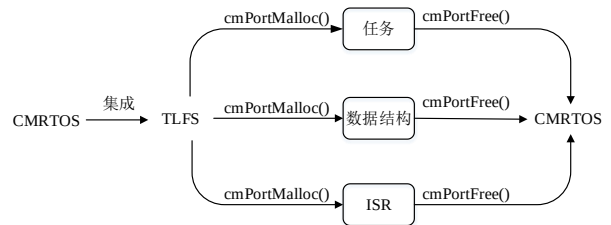


Fig. 4 Implementation process of memory management
图4 内存管理实现过程

2.2 时间管理模块

系统通过定时器来实现时间管理,既能确保系统时间的精确性及任务执行的时效性,还能在资源有限的情况下优化任务调度策略,提升系统整体性能与响应速度,从而支持嵌入式系统在各种复杂应用场景中高效、稳定地运行。CMRTOS系统提供了一组API来管理软、硬件定时器,以抽象和封装底层硬件定时器,使代码能在不同的Cortex-M微控制器平台上移植,无需修改即可运行应用程序。

CMRTOS操作系统基于Cortex-M处理器实现,使用SysTick定时器作为系统硬件定时器。SysTick定时器是一个支持中断的24位的向下计数定时器,被捆绑在嵌套向量中断控制器(Nested Vector Interrupt Controllers, NVIC)中产生SysTick异常。同时,该定时器作为ARM Cortex-M

内核处理器的标准组件,其普遍性和内置特性使其成为了CMRTOS操作系统中稳定系统时钟心跳的理想选择,增强了系统在不同Cortex-M系列硬件处理器平台上的兼容性和可移植性。

系统初始化阶段,详细配置和启动SysTick定时器,配置STK_CTRL、STK_VAL、STK_LOAD、STK_CALIB等寄存器,以特定频率递减计数,并在计数值达到零时产生SysTick中断。该中断机制为操作系统提供了稳定的时钟节拍,是系统时钟的核心源,能进行时间管理,保障任务调度精确执行。为了在满足系统实时性需求的同时,优化整体性能表现,CMRTOS将系统时钟节拍设定为5 ms,确保操作系统每5 ms精确触发一次SysTick中断,以响应系统任务调度、时间更新等关键操作,平衡系统的实时响应能力与整体运行效率。

CMRTOS的软件定时器不依赖于硬件,支持同时创建、管理多个定时器实例,与FreeRTOS软件定时器相同。其中,每个定时器实例均支持独立配置,包括定时周期、初始延迟以及与之关联的回调函数,以灵活适应不同任务的定时需求。同时,为了在满足系统性能的基础上确保实时性,将软件定时器的时间基准设置为10 ms,以适用于对时间精度要求不高的任务。

软件定时器支持单次模式、周期模式,可满足不同的应用需求。单次模式下,用户创建并启动定时器后,只执行一次回调函数后就会删除定时器,不再重新执行,也可被手动重启;周期模式下,定时器会按照设置的定时时间循环执行回调函数,直至用户将定时器删除^[19]。单次模式与周期模式之间的行为差异如图5所示。

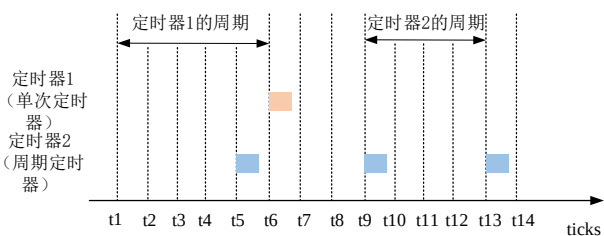


Fig. 5 Behavioral differences between single-time and periodic patterns

图5 单次模式与周期模式间的行为差异

2.3 任务调度模块

任务调度实现的基础是任务调度算法切换任务来成。CMRTOS基于优先级抢占式调度算法与相同优先级下轮询调度算法进行任务调度。其中,任务优先级通过数字表示,数字越小优先级越高,任务优先级可在任务创建时在任务控制块(Task Control Block, TCB)中设定^[20,21]。

TCB作为存储任务相关信息的数据结构,负责承载、维护包括任务优先级在内的多种元数据。在任务创建过程中,任务调度器会将这些优先级信息有效地插入全局优先级列表,确保系统按照既定优先级顺序进行调度与执行。TCB涉及的任务控制核心变量如表1所示。

Table 1 Task control block

表1 任务控制块

变量名称	变量含义
cmTopOfStack	指向任务栈顶的指针
cmPriority	任务优先级
cmStack	任务堆栈的起始指针
cmPID	任务ID
cmState	任务状态
cmEndOfStack	任务堆栈栈底指针

在CMRTOS中引起调度的时机包括:正在运行的任务进入阻塞状态、当一个任务的时间片用完、进行中断处理。在基于优先级的抢占式调度机制时,系统遵循一系列步骤以确保任务切换的高效性与系统运行的稳定性。

首先,事件触发调度后,系统会立即对当前正在执行的任务执行状态进行快照操作,保存处理器寄存器值、程序计数器(PC)、堆栈指针(SP)等关键的上下文信息,旨在为后续任务恢复提供一个精确无误的起始点,确保任务能无缝接续至中断前的执行状态;其次,调度器根据优先级抢占式调度策略评估、筛选当前就绪任务队列中的优先级最高者,作为下一个执行目标;最后,系统着手恢复被选中任务的上下文环境,即重新加载保存的状态信息,并传输至相应的硬件寄存器与内存结构中,包括但不限于前述的寄存器值、PC及SP等。此步骤的精确执行是保障多任务并发执行时系统稳定性和一致性的关键。

随着新任务上下文完全恢复,CPU控制权顺利交接至该任务,标志着任务调度的完成与新任务执行的开始或继续。同时,调度器同步更新其内部管理的数据结构,例如任务队列与优先级队列,以精确反映系统当前任务状态及资源分配情况,为后续调度决策提供实时、准确的信息,便于后续进行基础性能测试与结果分析。具体流程如图6所示。

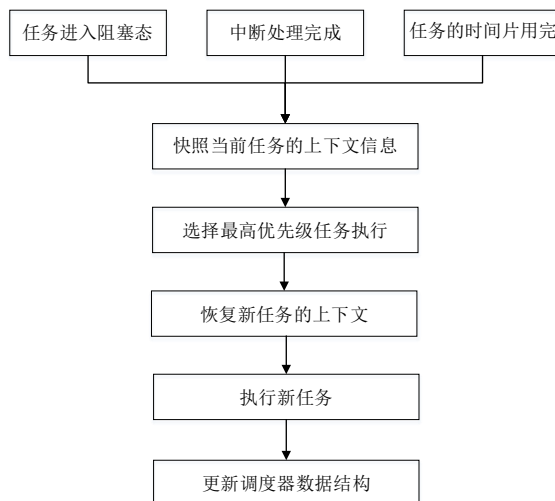


Fig. 6 Task scheduling process

图6 任务调度流程

2.4 通信管理模块

通信管理主要用于任务间的数据传输、同步、协调等,确保多任务应用能在高实时性场景中高效、可靠地运行。为了更好地应用资源受限的嵌入式系统,除了消息队列、信号量和事件组外,CMRTOS 还增加了一种任务直接通知的通信机制,适用于任务间、从中断服务程序(Interrupt Service Routine, ISR)到任务间的简单信号与数据传输,在不大幅增加内存负担的前提下,为实现系统内的定时任务同步与通信提供了一个高效、可行的方案。

任务直接通知机制为每个任务分配一个轻量级的、专用的 32 位通知值,表示不同状态、计数或信号,也可用于传递简单数据。该值存储在任务的 TCB 中,不依赖复杂的数据结构,显著减少了于队列、信号量和事件组等操作^[22]。具体而言,每个任务的 TCB 包含一个 cmNotifiedValue 字段保存任务通知值,可直接读取或修改自己的通知值。同时,通过 cmTaskNotify() 函数向指定的任务发送带值的通知,cmTaskNotifyFromISR() 函数则用来在中断服务例程中发送通知,cmTaskNotifyTake() 函数负责接收通知并清除通知值。此外,cmTaskNotifyGive() 函数提供了一种发送简单通知的方法,通常用于设置标志位。

该机制的优势在于:具有直接性与高效性,能在不创建额外复杂结构体的情况下,直接接收来自其他任务或 ISR 的通知,有效减少通信过程中的中间环节与数据封装开销,提升通信效率。任务直接通知实现流程如图 7 所示。

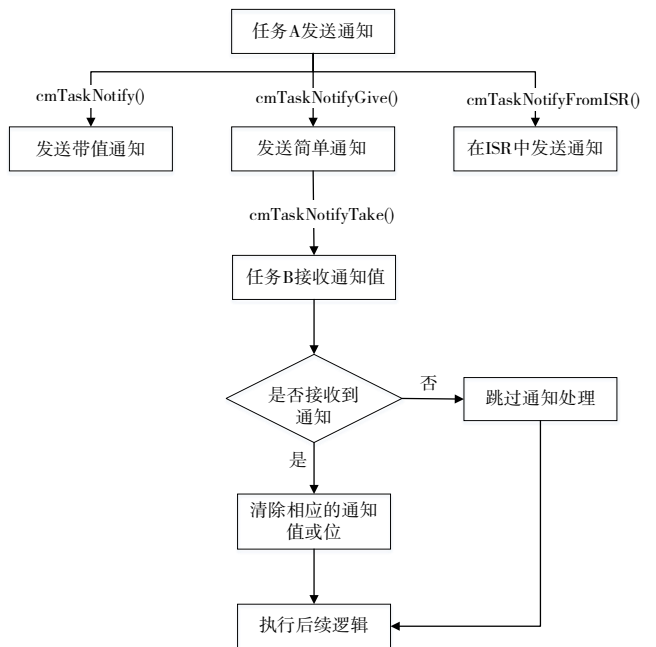


Fig. 7 Task direct notification flow
图7 任务直接通知流程

由于任务直接通知只能传递一个 32 位值,无法传递复杂的数据结构,因此更适用简单的同步或数据传递场景。为此,可将任务直接通知机制与其他通信机制(消息

队列、信号量、事件组)相结合,以构建一个更灵活、全面的通信体系,满足多样化应用场景下对高效数据传递、精准同步及快速信号响应的严苛需求。

3 系统性能测试

本文针对嵌入式操作系统的需求,对 CMRTOS 系统的实时性进行测试与分析。首先,执行一项启动测试流程获得测试结果;其次,对测试结果进行分析,旨在确保操作系统能在预期环境中正确、稳定地进行初始化与运行。

3.1 测试环境

测试硬件平台选择东莞野火电子技术有限公司的野火霸天虎 V1 开发板,搭载了高性能的 STM32F407 处理器,内核为 Cortex-M,处理器支持高达 168 MHz 的时钟频率,Flash 存储器最大为 1 MB,SRAM 为 192 KB,将 fire debugger 全速版作为测试所用仿真器。

fire debugger 全速版作为一个人体学接口设备,支持所有基于 Cortex-M 内核的单片机,常见的 M3、M4 和 M7 都能完美支持,无需繁琐的驱动安装过程即可直接使用。此外,该版本支持 Keil 开发环境,可采用 SW 下载模式直接下载代码,具备在线调试与硬件仿真的能力,测试系统连接图如图 8 所示。

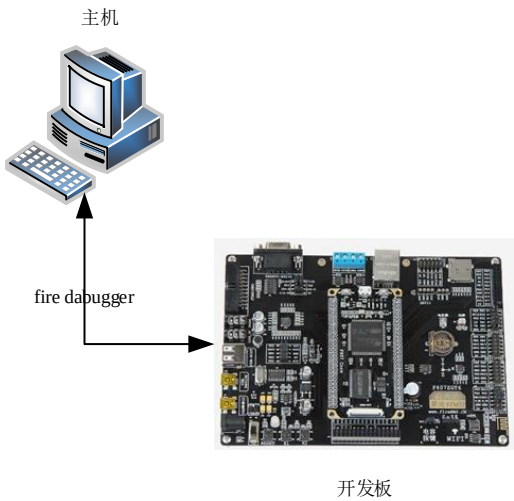


Fig. 8 Test system
图8 测试系统

3.2 启动测试

本文对系统进行启动测试的目的是保证系统实验平台可用,即将系统移植到开发板上,编写一个多任务程序的测试用例进行测试,主要观察野火多功能调试助手软件的串口打印结果、开发板状态变化。测试程序流程如图 9 所示。

具体为,编写好程序后对程序进行编译,将程序通过仿真器 fire debugger 烧录到开发板上,运行 3 个测试任务:任务 1 优先级为 2,每间隔 500 ms 闪烁一次 LED 灯;任务 2

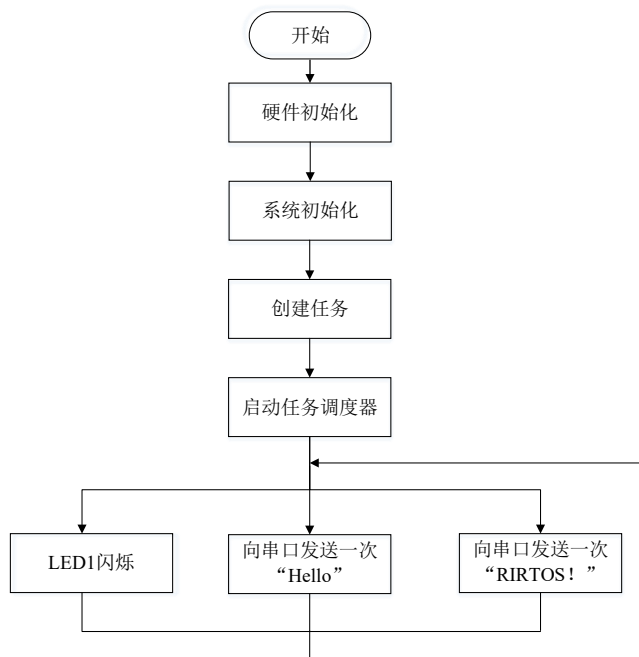


Fig. 9 Process of running test program

图9 运行测试程序流程

优先级为3,每间隔2 000 ms向串口发送一次“Hello”;任务3优先级为3,每间隔5 000 ms向串口发送一次“RIRTOS”。由此可见,开发板上LED灯周期闪烁,串口数据通信正常,如图10所示。

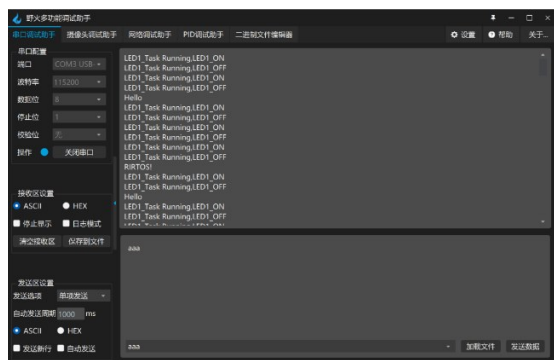


Fig. 10 Serial port test results

图10 串口测试效果

CMRTOS的任务调度机制基于优先级的抢占式调度,任务1只有在任务2、任务3处于等待状态时才能执行。观察LED灯闪烁周期和串口助手软件接收的信息可见,测试结果与预期结果相符,任务调度结果符合CMRTOS系统的任务调度原理,串口通信也正常。

3.3 内存利用率测试

一个高效的内存分配器应充分利用有限的内存资源,最大限度地减少内存碎片的产生。内存碎片通常可分为内部碎片与外部碎片:内部碎片是指分配的内存块中未被实际使用的部分,即占用的资源未能被有效利用;外部碎片是指虽然有足够的内存总量,但因空闲内存块分散,无法满足较大内存块的分配需求。二者数量越少,内存利用率就越高,减少内存碎片是提升内存利用效率的关键。

本文采用动态内存基准测试程序,通过分解给定的数字因数实现多次内存分配与释放请求。实验平台选用STM32F407开发板,嵌入式操作系统为FreeRTOS,堆内存大小设定为64 KB(65 536 B)。统计发现,内存初始化过程内存消耗为13 B,堆内存剩余大小为65 523 B,通过相关堆状态函数可知堆内存中主要的信息包括:内存总大小Total、所有空闲内存块之和大小Allfree、最大空闲内存块大小Maxfree、已使用内存块大小Used。根据式(1)、式(2)可计算出内部碎片率与外部碎片率^[23]。

$$\text{内部碎片率} = \frac{\text{Used}}{\text{Total}} * 100\% \quad (1)$$

$$\text{外部碎片率} = (1 - \frac{\text{Maxfree}}{\text{Allfree}}) * 100\% \quad (2)$$

由于FreeRTOS默认采用heap_4提供的First-Fit作为内存分配策略,将重点比较Good-fit策略与First-Fit策略的性能差异。为了确保实验结果的可靠性,每种内存分配策略测试均进行20次实验并取平均值,结果如表2所示。由此可知,采用Good-fit策略的内存管理相较于First-Fit策略减少了6%的内存内部碎片率,节省了约3 931 B的内存空间。进一步分析表明,TLSF算法采用的Good-fit策略通过更精细的内存块分配与管理,显著降低了内存碎片率和内存空间,证明了将TLSF算法引入CMRTOS中对内存管理的性能优于FreeRTOS,可有效减少内存碎片,提升内存利用效率。

Table 2 Comparison of memory allocation strategies

表2 内存分配策略结果比较

内存分配策略	Total/B	Allfree/B	Maxfree/B	Used/B	内部碎片率/%	外部碎片率/%
First-Fit	65 523	6 180	2 471	58 819	89.77	60.02
Good-fit	65 523	9 313	2 471	54 679	83.45	73.47

3.4 实时性测试

本文采用软件调试功能的硬件仿真方案测试系统实时性,通过Keil uVision5软件DEBUG模式下内置的周期计时器,测量不同代码的具体运行时间。首先,在需要测试的代码段两端设置断点并记录时间Sec1、Sec2;其次,进入DEBUG模式全速运行,得到每个过程所花费的时间ΔSec。

$$\Delta \text{Sec} = \text{Sec2} - \text{Sec1} \quad (3)$$

式中:ΔSec为目标程序运行时间,单位为s。

软件调试功能是基于DAP硬件仿真的方式,需将硬件仿真器频率与目标处理器频率设置一致,才能保证测试结果的准确性。由于测试使用的STM32F407的时钟频率为168 MHz,因此要将仿真器的核心频率设置为168 MHz才能得到实际运行时间。

为了增强实验结果的可靠性与精确度,采取了多次重复测量的策略。具体为,实验针对每项指标均执行100次独立测试,并取平均值作为最终评估依据,以削弱偶然误差对结论的影响。图11为在统一的软硬件环境框架下,比较CMRTOS与FreeRTOS嵌入式操作系统的关键实时性

能指标。由此可见,CMRTOS在3项指标上优势明显,不仅实时性良好,还体现出了更高的稳定性。

表3为两种操作系统在相同条件下,针对上述3项实时性指标进行100次测试所得的平均耗时。由此可知,在保持外界因素一致的前提下,CMRTOS在任务上下文切换、任务调度、任务响应方面的耗时稳定在5 us以内,验证

了所提方法在实时性方面的有效性与可靠性。

Table 3 Comparison of real-time index test results between CMRTOS and FreeRTOS

测试项	测试时间/us	
	FreeRTOS	CMRTOS
上下文切换	2.46	0.75
任务调度	7.16	2.73
任务响应	10.54	4.51

4 结语

本文基于Cortex-M架构,在FreeRTOS基础上设计、开发了一种嵌入式实时操作系统CMRTOS。通过集成TLSF算法、调整任务调度方式及采用轻量级通信机制,对FreeRTOS操作系统进行优化,使其更适用于高实时性需求及资源受限的设备环境。实验表明,CMRTOS在实时性、稳定性和内存管理效率上,相较于FreeRTOS提升明显。

然而,CMRTOS基于Cortex-M架构所设计,在大规模应用中会面临跨平台开发的挑战,且系统尚未开源。未来,将集中于构建一套高度抽象化、一致性的API接口体系,实现操作系统在不同硬件平台间的无缝移植与代码高效复用;将继续优化系统性能,在完成最终验证后公开源代码,促进代码不断更新迭代和开源社区的发展。

参考文献:

[1] AHMAD A, ANISETTI M, DAMIANI E, et al. Special issue on real-time image and video processing in mobile embedded systems[J]. Journal of Real-Time Image Processing, 2019, 16(1): 1-4.

[2] SANGIOVANNI A, DI N M, LUCIA L B, et al. Recent advances and trends in on-board embedded and networked automotive systems [J]. IEEE Transactions on Industrial Informatics, 2019, 15(2): 1038-1051.

[3] KIRANKUMAR P, VASANT G, ARVIND D. Smart home automation system using advanced embedded system platforms with AI and IoT[J]. International Journal for Research Trends and Innovation, 2022, 7 (7) : 328-333.

[4] MALINOWSKI A, YU H. Comparison of embedded system design for industrial applications [J]. IEEE Transactions on Industrial Informatics, 2021, 7(2): 244-254.

[5] GUAN F, PENG L, PERNEEL L, et al. Open source FreeRTOS as a case study in real-time operating system evolution [J]. Journal of Systems and Software, 2016, 118: 19-35.

[6] GONG X H, JIANG B Z, CHEN X L, et al. Survey of real-time computer system architecture [J]. Journal of Computer Research and Development, 2023, 60(5): 1021-1036.

龚小航, 蒋滨泽, 陈香兰, 等. 实时计算机系统结构综述[J]. 计算机研究与发展, 2023, 60(5): 1021-1036.

[7] KARLA S, IVAN P, HRVOJE D. Dynamic priority assignment in FreeRTOS kernel for improving performance metrics [C]// Opatija: International Convention on Information, Communication and Electronic Technology,

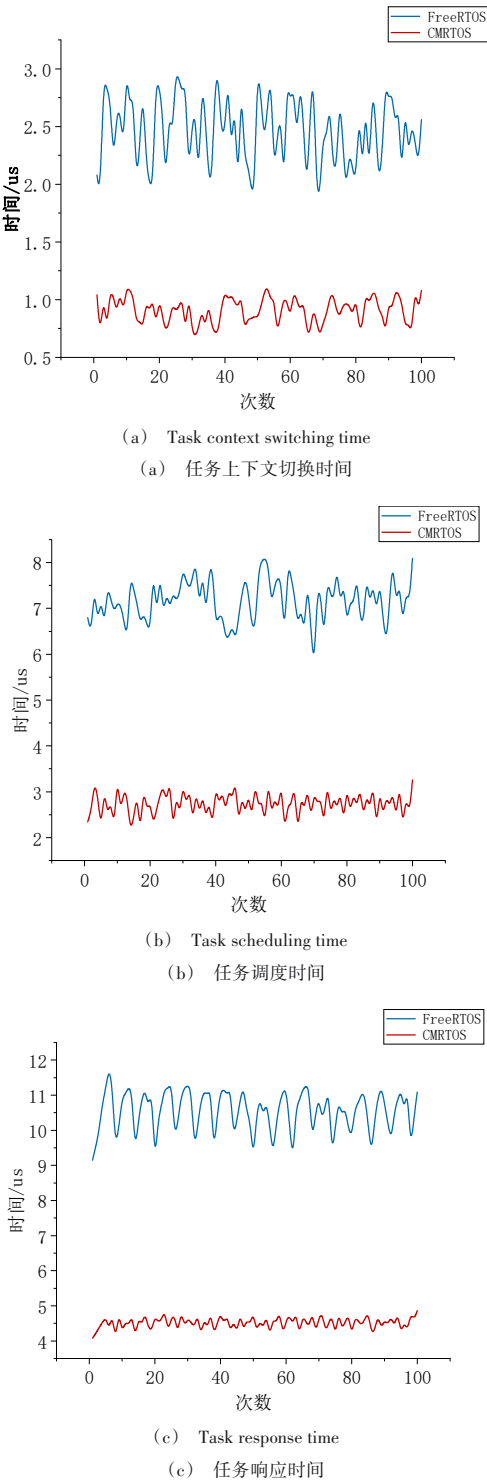


Fig. 11 Time taken for real-time performance metrics of CMRTOS and FreeRTOS

图 11 CMRTOS 与 FreeRTOS 实时性指标所用时间

- 2021.
- [8] CHUNMAO J, GUOYIN Z, CHUNMEI H. Research of embedded operating system based on multi-core processor[C]// Chengdu: 3rd International Conference on Computer Science and Information Technology, 2010.
- [9] HEMANTHKUMAR P B, SHREEKAR R A, JOSH F T, et al. Introduction to ARM processors & its types and overview to Cortex-M series with deep explanation of each of the processors in this family[C]// India: International Conference on Computer Communication and Informatics, 2022.
- [10] ANONYMOUS. STM32F405xx STM32F407xx [EB/OL]. <https://www.st.com/resource/en/datasheet/stm32f405rg.pdf>.
- [11] EMMANUEL B, CENK G, OLIVER H, et al. RIOT: an open source operating system for low-end embedded devices in the IoT[J]. IEEE Internet of Things Journal, 2018, 5(6): 4428–4440.
- [12] CHEN X N. Design and research of embedded operating system based on Cortex-M4[D]. Nanjing: Nanjing University, 2019.
陈相宇. 基于Cortex-M4的嵌入式操作系统设计研究[D]. 南京: 南京大学, 2019.
- [13] ANONYMOUS. STM32F3 and STM32F4 series Cortex-M4 programming manual [EB/OL]. <https://www.slideshare.net/slideshow/manual-programmao-stm-32-f4/58959507>.
- [14] CAI W T, WEN H S, BEADLE H A, et al. Understanding and optimizing persistent memory allocation[C]// London: ACM SIGPLAN International Symposium on Memory Management, 2020.
- [15] EETU A. Embedded dynamic memory allocator optimization[D]. Finland: Tampere University, 2022.
- [16] LIU H L, YANG S. FreeRTOS kernel implementation and application development practical guide[M]. Beijing: Mechanical Industry Press, 2019.
刘火良, 杨森. FreeRTOS内核实现与应用开发实战指南[M]. 北京: 机械工业出版社, 2019.
- [17] SANIDHYA K, CHANGWOO M, TAESOO K. Scalable NUMA-aware blocking synchronization primitives [EB/OL]. <https://www.usenix.org/system/files/conference/atc17/atc17-kashyap.pdf>.
- [18] REYNOLDS J H. Using bit manipulation to reduce sequential search times[J]. Journal of Computing Sciences in Colleges, 2001, 17(2): 263–270.
- [19] YAO J L, WANG S L, WANG H. Function exploration of real-time operating system FreeRTOS+CANopen[J]. Integrated Circuits and Embedded Systems, 2019, 19(9): 22–25.
姚家亮, 王树磊, 王昊. 实时操作系统FreeRTOS+CANopen功能探索[J]. 集成电路与嵌入式系统, 2019, 19(9): 22–25.
- [20] CHANDIRAMANI K, VERMA R, SIVAGAMI M. A modified priority preemptive algorithm for CPU scheduling[J]. Procedia Computer Science, 2019, 165: 363–369.
- [21] MICRIUM. μ C/OS-III users manual [EB/OL]. <https://www.analog.com/media/en/dsp-documentation/software-manuals/Micrium-uCOS-III-UsersManual.pdf>.
- [22] SILBERSCHATZ A, GAGNE G, GALVIN P B. Concept of operating system[M]. Translated by Zheng kougen, Tang Jie, Li Shanping. Beijing: China Machine Press, 2020.
SILBERSCHATZ A, GAGNE G, GALVIN P B. 操作系统概念[M]. 郑扣根, 唐杰, 李善平, 译. 北京: 机械工业出版社, 2020.
- [23] WANG Z W, JIANG Z J, CHEN J C. A design scheme for improving real-time property of memory management in Linux[J]. Computer Engineering, 2014, 40(9): 291–294, 299.
王兆文, 蒋泽军, 陈进朝. 一种提高Linux内存管理实时性的设计方案[J]. 计算机工程, 2014, 40(9): 291–294, 299.

(责任编辑: 刘嘉文)