

嵌入式多核系统的可调优先级自旋锁

原创 王月, 李杰 嵌入式系统专家之声 2025年08月20日 08:00

[点击上方"蓝字"](#) [关注我们吧!](#)



嵌入式系统专家之声

嵌入式系统专家之声依托嵌入式系统联谊会专家团队, 汇集嵌入式系统产、学、研和媒体...
199篇原创内容

公众号

2025.08.20

本文字数: 5572

预计阅读时长: 14分钟

摘要

传统自旋锁具有无序竞争的特点, 本文对自旋锁机制进行了研究, 通过引入优先级和等待次数阈值提出可调优先级自旋锁, 保证高优先级任务尽可能多地获得锁, 低优先级任务经过一段等待时间后调整优先级、增加获取锁的机会, 实现了可调优先级自旋锁并通过实验进行了验证。测试结果表明, 可调优先级自旋锁既能够减少传统自旋锁的时间开销, 又能保证高优先级处理器核锁申请较快得到响应, 验证了可调优先级自旋锁在多核系统中是可用的。

关键词

关键词: 多核; 互斥; 自旋锁; 优先级

作者: 王月, 李杰

中图分类号: TP311

文献标识码: A

0 引言

多核处理器中各核共享系统中硬件资源, 共享内核程序和内核数据等核间临界资源^[1], 当多个处理器核同时访问共享资源时, 会产生访问冲突问题。为保证多核间临界资源的同步互斥访问, 通常采用信号量、读写锁、自旋锁等方法。信号量是一种睡眠锁, 当信号量不可用时会使任务睡眠, 处理器可进行其他工作, 适用于占锁时间较长的情况^[2]。读写锁在互斥访问中, 可以保证同一时刻一个写者占有写锁, 多个读者并发读^[3]。自旋锁则保证只有一个处理器核使用临界资源。当自旋锁已被占用时, 其他处理器核自旋检测锁是否被释放, 至锁空闲才可获取。通常情况下, 自旋锁的占

有时间短，没有上下文切换的开销，锁被释放后可以较快地获得，效率相对较高。相关研究表明，自旋锁是多核系统中的一项关键技术^[1]。自旋锁虽保证了临界资源的互斥访问，但无法保证访问临界资源的处理器核顺序，导致多核间任务执行顺序随机等不确定性问题。本文针对嵌入式系统中任务重要程度不同的问题，提出可调优先级自旋锁，并通过测试验证了在多核系统中的可用性。

1 自旋锁机制研究

1.1 锁基本原理

锁的实现主要考虑两方面：一是与锁相关的指令，二是处理器核检查锁时其状态的可见性。锁是通过系统内存中特定位置的一块数据来实现的^[2]。锁的实现需要处理器原语指令配合，大多数处理器都提供test - and - set原语指令。另一方面，当锁状态发生改变时，该变化应对所有正在运行的处理器核可见。处理器的高速缓存实现write -through模式，占用锁的处理器核写入到高速缓存的数据也会使内存中的对应数据被同步更新^[4]，并通过处理器硬件总线协议保证高速缓存的一致性，使其他申请锁的处理器核读取到修改后的最新状态^[5]。

1.2 传统自旋锁

传统自旋锁维护一个锁变量值flag，进行初始化时将锁变量值初始化为data1，表示处于空闲状态。进行加锁时，处理器核判断锁变量值，当抢到锁时修改锁变量值为data2，表示自旋锁正在被占用，其他处理器核则自旋等待。由于处理器核获取锁的顺序随机，获取锁的时机不确定，因此任务的响应时间延迟无法确定^[6]。

1.3 确定自旋锁

为缓解传统自旋锁的不确定性，许多研究提出改进自旋锁，使自旋锁的获取顺序按照先来先服务的原则，如排队自旋锁是模拟一条先进先出的队列，维护内存中next和owner变量^[7]。MCS Lock是一种基于链表的可扩展、公平自旋锁，使用队列来维护申请者的顺序^[8]。对其改进后有HMCS (Hierarchical MCS)锁^[9]和HCMS-T锁^[10]。CLH Lock也是一种基于单链表的公平自旋锁，但在其前驱接节点本地变量上自旋^[11]，其后出现了HCLH Lock (Hierarchical CLH Lock)^[12]。

自旋锁机制可以解决自旋锁“不公平”问题，适用于各个处理器核上任务需求无差别情况。但对嵌入式系统而言，通常对系统中某些任务的响应时间有一定的要求，各处理器核任务存在优先次序之分，而锁的获取顺序对系统中任务的响应时间造成影响，上述自旋锁机制不能满足紧急任务优先获得锁的要求。

2 可调优先级自旋锁

2.1 基本思路

对不同任务锁申请设置不同优先级，从而控制获取锁的顺序，保证在申请锁队列中，优先级高的任务可优先获取自旋锁。本文假设在任务持有锁期间不会被中断，更高优先级锁申请只有在锁释放后才能获取锁。

在固定优先级的情况下，当高优先级任务较多时，会出现低优先级的任务无法抢到锁、相关任务无法执行的问题，即出现饥饿现象。为缓解这一问题，当某些处理器核超过一定等待时间后提高

其优先级，增加该处理器核获取锁的机会。

2.2 改进措施

2.2.1 固定优先级

引入优先级措施，实现按优先次序获取自旋锁。假设共有 n 个处理器核，若 i 为处理器核编号，则 $i \in [0, n-1]$ 。处理器核 i 的优先级为 $P_i = n-i$ ，其优先级顺序为 $P_0 > P_1 > P_2 > \dots > P_{n-1}$ 。若当前申请锁的处理器核的集合记为 CP ， $CP = \{P_i, P_j, \dots, P_k\}$ ，获得锁的处理器核由 $P_m = \text{Max}(CP)$ 确定，则集合中最大值 P_m 对应的处理器核 m 成功获取锁。

各处理器核锁申请优先级固定，但会导致低优先级处理器核长时间无法获取锁。

2.2.2 引入等待次数阈值

为增加低优先级处理器核获取锁的机会，考虑到临界区操作时间的长短对等待时间造成影响，选择根据等待次数设置阈值，即当高优先级处理器核抢锁次数超过设定阈值时，调整等待序列中尚未有机会获得锁的处理器核的优先级。

当系统中处理器核数较多时，若对所有优先级设置不同的阈值进行调整，会使处理器核获取锁的顺序较为复杂。为了简化策略，使获取锁的顺序更易控制，选择对优先级进行分层调整，同一层的处理器核设置相同的阈值。若每层中包含 m 个处理器核，则共有 $\lceil \frac{n}{m} \rceil$ 层，处理器核 i 的优先级 P_i 如式（1）所示：

$$P_i = \begin{cases} n-i & t \leq \text{threshold} \\ n-i+n & t > \text{threshold} \end{cases} \quad (1)$$

其中， threshold 为根据需要选择的阈值， t 为该处理器核的等待次数。当申请锁的处理器核等待次数超过阈值时，该处理器核优先级提高 n 级，则此时优先级为 $P_i = 2n-i$ 。

通过调整处理器核优先级，低优先级处理器核能够在超过阈值后提高优先级，提高其获得锁的机会，缓解固定优先级引起的饥饿现象。

2.3 实现流程

可调优先级自旋锁加锁流程以4核处理器为例，其中处理器核0的固定优先级 $P_0=4$ ，其固定优先级最高。将4种不同优先级处理器核分为两层，处理器核0和1为第一层，处理器核2和3为第二层。

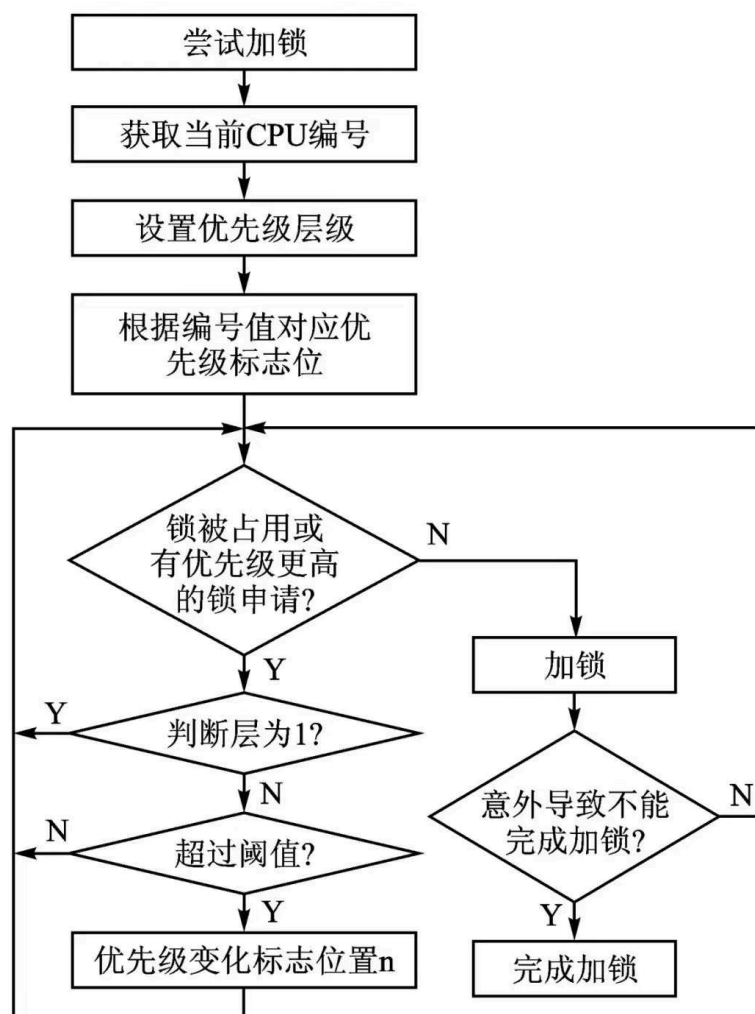


图1 动态优先级加锁流程

图1显示其加锁基本流程。可调优先级自旋锁加锁流程为：①获取当前申请锁处理器核编号cpuid；②设置其优先级层级，处理器核0和核1层级为1，处理器核2和核3层级为2；③根据处理器核编号cpuid将对应优先级标志置1，表示有对应优先级的锁申请；④判断锁是否被占用以及等待队列中是否有比自身优先级更高的锁申请；⑤若锁空闲并且无高优先级锁申请，则使用ldstub完成加锁操作，否则自旋等待；⑥自旋过程中首先判断层级，若层级为1，则执行步骤④，不进行优先级调整；⑦若层级为2，判断占有锁处理器核发生变化次数是否超过阈值，若不超过阈值，则继续自旋；⑧若处理器核等待占有锁处理器核发生变化的次数超过阈值，则将level数组中对应数据置n，提高其优先级，跳转至步骤④操作；⑨判断是否加锁成功，加锁成功退出程序，加锁失败则跳转步骤④，操作重新尝试加锁至成功。

解锁操作步骤为：①获取当前申请锁的处理器编号cpuid；②根据得到的处理器核编号cpuid将对应所有标志变量清0。

3 实验

3.1 实验环境

使用Xilinx四核Leon3 SPARC V8处理器开发板，在无操作系统、主频为100 MHz的环境中，将裸机程序下载并加载到开发板运行。实验在4个处理器核中并发执行，不涉及线程并发执行。

3.2 实验方法

实验中将任务与处理器核绑定，并保证各个处理器核实现相同的抢占锁功能。实验进行3组测试。测试实验1：统计传统自旋锁，固定和可调优先级自旋锁各处理器核在规定时间内抢占锁次数。测试实验2：测试调节阈值对各处理器核抢占锁次数的结果影响。测试实验3：对比上述3种自旋锁机制无竞争获取锁延迟以及有竞争情况下总运行时间，分析时间开销大小。每个实验运行50次，最终结果取平均值。

3.3 实验结果分析

测试实验1中各处理器核临界区操作时间大约为9000 μ s，各自旋锁机制抢占锁次数测试结果如图2所示。使用传统自旋锁时，各核抢锁次数差距较小。在固定优先级时，处理器核0和核1优先级较高，两者交替频繁获得锁，而核2、核3在核0与核1都工作的情况下无法抢到锁。对于可调优先级自旋锁（阈值为6），核2和核3在锁被高优先级核获取6次后有机会获得自旋锁。相比传统自旋锁，可调优先级中核0和核1抢占锁次数增加49.8%和49.4%，同时核2和核3降低50.2%和50.1%。相比固定优先级，可调优先级自旋锁虽牺牲高优先级抢占次数，但提高了低优先级获取锁的机会，缓解了饥饿问题。

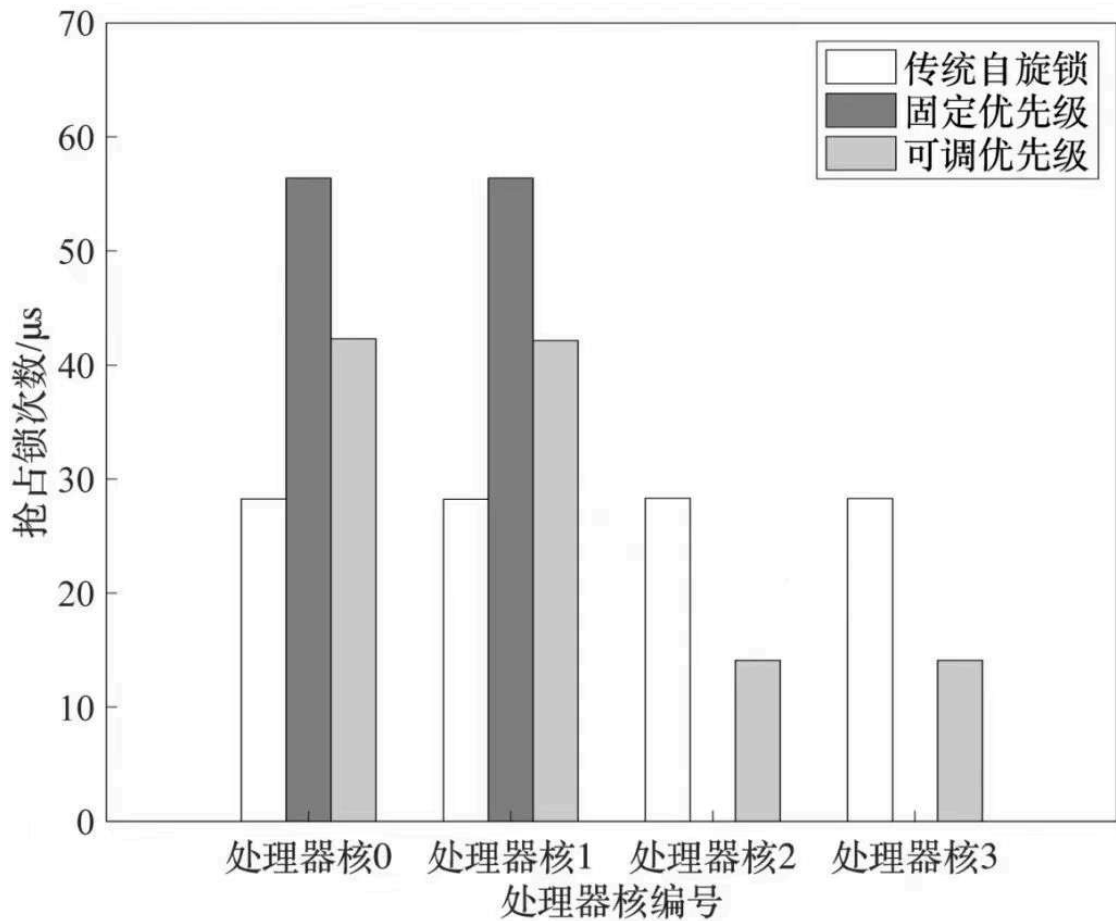


图2 不同优先级抢占锁能力测试结果

临界区操作时间的长短影响每秒内抢占锁的次数，但不改变其变化趋势，测试实验2的结果如图3所示。当阈值增大时，核2和核3在等待较长时间后才可获得锁，抢锁次数随着阈值的增大而减小，核0和核1抢锁次数则随阈值增大而增大。当阈值为2时，各个处理器核抢占锁次数相等，其效果相当于传统自旋锁；当阈值增大到大于处理器核0和核1执行所有任务完时，效果等效于固定优先级。

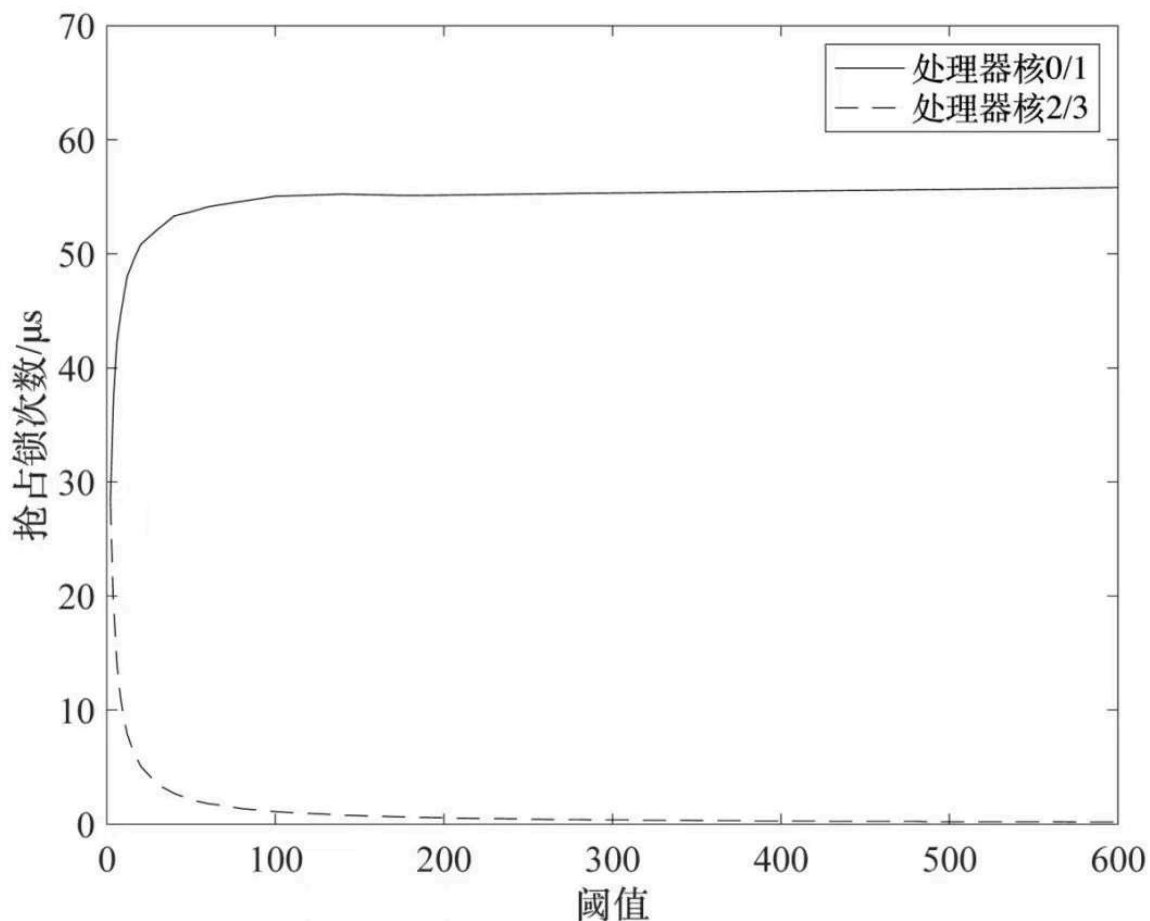


图3 调整阈值测试结果

测试实验3中传统自旋锁与固定优先级和可调优先级自旋锁的无竞争获取锁延迟测试结果如表1所列，其平均值依次为 $1.08\mu\text{s}$ 、 $1.58\mu\text{s}$ 、 $2.01\mu\text{s}$ 。引入优先级的自旋锁会增加一定的时间开销，但对程序性能造成影响相对较小。

表1 无竞争抢锁程序运行时间比较

算法	次数/次		
	200	300	500
传统自旋锁	212	325	541
固定优先级	316	467	793
可调优先级	402	604	999

测试实验3中有竞争获取锁总程序的时间开销如图4所示。可以看出，传统自旋锁由于不确定性造成冲突较多，导致时间开销最大。而固定优先级自旋锁中消除了加锁过程中的冲突，总时间开销降低，平均每次加解锁操作减少 61.08 ps 。可调优先级自旋锁由于相对于固定方式时间和空间复杂度更高，平均每次加解锁操作增加 $22.23\mu\text{s}$ ，但对比传统自旋锁减少 $38.85\mu\text{s}$ 。

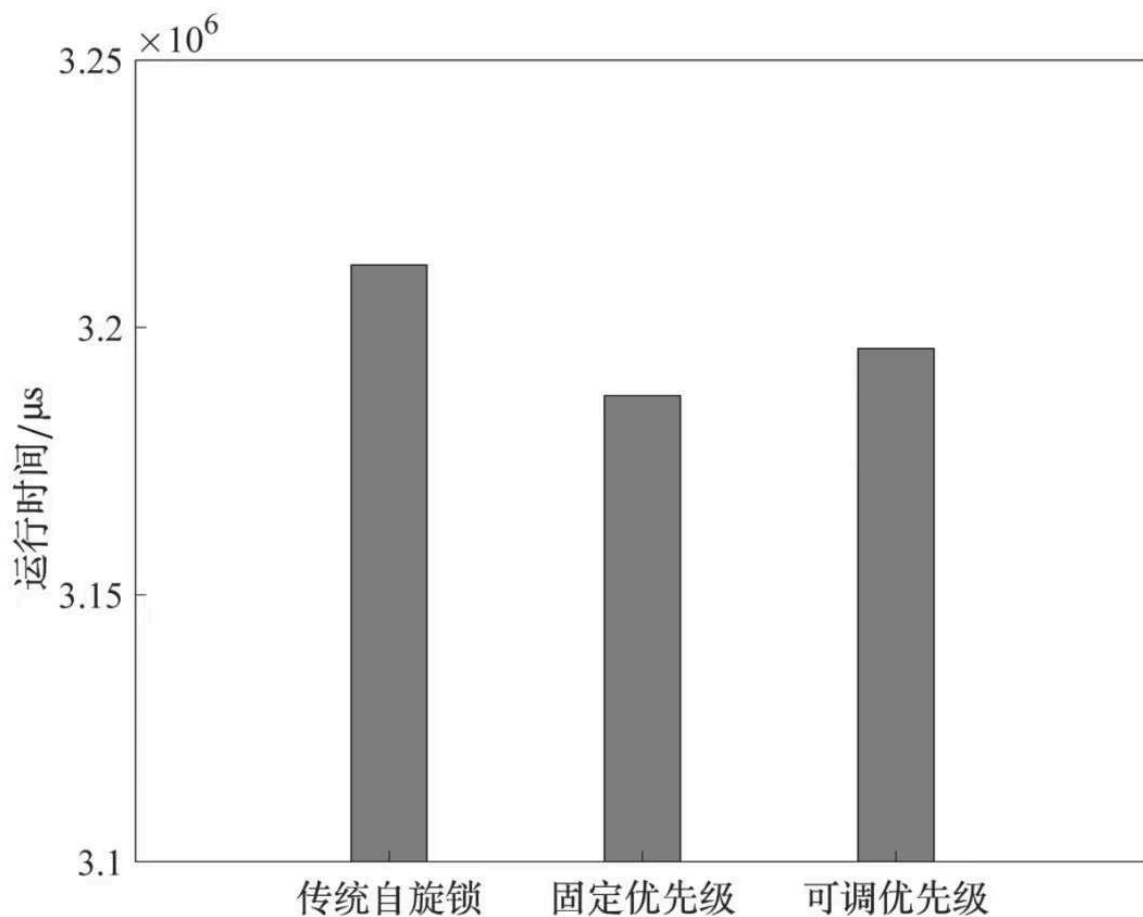


图4 有竞争抢占锁时间开销测试结果

4 结语

目前大多数多核系统均可以通过自旋锁机制实现共享资源的互斥访问，但传统自旋锁获取锁顺序不确定，无法满足某些嵌入式环境紧急任务需要及时响应的情况。本文提出并实现可调优先级自旋锁，既确保高优先级任务尽可能优先获得自旋锁，又可以使低优先级任务在经过一段等待时间后调整优先级、增加获得锁的机会。对可调优先级自旋锁的各处理器核抢占锁次数、阈值影响和时间开销进行了测试分析。测试结果表明，可调优先级自旋锁能在引入开销不大的情况下实现预期目标，且可根据需要对等待阈值进行调整，以满足不同系统需求，在任务重要程度不等的多核环境中是可行的。

参考文献

.....

- [1]虞保忠, 郝继锋. 多核操作系统自旋锁技术研究[J]. 航空计算技术, 2017,47(4):115-117.
- [2]李畅. 信号量实现进程同步与互斥过程详细分析[J]. 计算机产品与流通, 2019(2):157.
- [3]张恒, 陈海波. 一种检测竞争并自调节的动态读写锁[J]. 小型微型计算机系统, 2016, 37(9):1904-1909
- [4]Gang H,Zeng H,M D Natale,et al. Experimental Evaluation and Selection of Data Consistency Mechanisms for Hard Real-Time Applications on Multicore Platforms [J]. IEEE Transactions on Industrial Informatics,2014,10(2):903-918.
- [5]Vijay Nagesan. A Primer on Memory Consistency and Cache Coherence[M]. San

(作者单位: 山东航天电子技术研究所, 烟台 264003)

(本文由《单片机与嵌入式系统应用》杂志社授权发表, 原文刊发在2021年第12期)

----- END -----

 嵌入式系统专家之声推荐搜索

嵌入式系统 | 物联网 | 嵌入式软件


【点击上方  搜索词条可查看号内更多其他内容】






微信搜一搜

 嵌入式系统专家之声

 ----- 关注我们, 了解更多精彩内容 ----- 

 转发, 点赞, 在看, 安排一下?

期刊论文 · 目录

上一篇 · RISC-V 向量指令集的 Yolov3 移植优化

留言

写留言

