

Linux 实时调度算法研究

吴振亚¹, 李 航², 苏锐丹¹

(1. 西安电子科技大学 计算机学院, 陕西 西安 710071;
2. 西安电子科技大学 软件学院, 陕西 西安 710071)

摘 要:随着 Linux 的普及, Linux 的实时性越来越受重视, 但目前对该领域内研究进行综述的工作较少, 并且已有的综述性工作主要从算法的自身结构对实时调度算法进行划分, 而非从使用者的角度, 根据解决的问题模式去划分, 从而不易于指导工业级的应用实践。为此, 文中从 Linux 实时调度算法所针对的问题模式出发, 对实时算法进行分类。分类基于由内到外, 从通用问题场景到具体应用问题场景的原则, 最终形成了层次良好的分类结构。该分类的优点在于: 除了更易于指导工业级的应用实践, 也更加准确地描述了实时调度算法的本质特征。

关键词:Linux; 实时; 调度算法; 问题模式

中图分类号:TP316.2

文献标识码:A

文章编号:1673-629X(2015)02-0033-05

doi:10.3969/j.issn.1673-629X.2015.02.008

Research on Linux Real-time Scheduling Algorithm

WU Zhen-ya¹, LI Hang², SU Rui-dan¹

(1. School of Computer, Xidian University, Xi'an 710071, China;
2. School of Software, Xidian University, Xi'an 710071, China)

Abstract: With the popularity of Linux, the real-time features of Linux have attracted more attention, but there is a little research survey in this field at present, and existed survey mainly classifies real-time scheduling algorithm according to structure of algorithm rather, than from the user's view, classifying according to the problem pattern. So, it's not easy to guide the practice in industry by this way. For this reason, classify the real-time scheduling algorithm according to the problem pattern. The principles that the classification follows are from inside to outside, from general problem scenario to specific problem scenario. Finally, a well-organized hierarchy is obtained. The benefits of this classification are that besides guiding the practice of industry more easily, it describes the nature of real-time scheduling algorithm more accurately.

Key words: Linux; real-time; scheduling algorithm; question pattern

0 引言

随着 Linux 的普及, 特别是 Linux 在日益广泛的智能终端上的应用, Linux 的实时性越来越受到重视, 但目前对该领域内研究进行综述的工作较少, 并且已有的综述性工作, 如文献[1-2]着重从算法的自身结构对实时调度算法进行划分, 而非从使用者的角度, 根据解决的问题模式去划分, 从而不易于指导工业级的应用实践。为此, 文中从 Linux 实时调度算法所针对的问题模式出发, 对实时算法进行分类。这样做的好处除了更易于指导工业级的应用实践, 也更加符合实时调度算法的本质特征: 所有的实时调度算法均和要

解决的问题的特性紧密相关, 而且即便是同一个问题, 也有可能存在不同的解决方案(这也是实时调度算法如此庞杂的根本原因)。此外, 特别需要指出的是: 文中仅涉及那些均已在 Linux 上实现的实时调度算法, 而不是理论上设计或没有在 Linux 上进行实现的实时调度算法。

文中采用的分类方法是: 首先将这些问题模式分为两类:

(1) Linux 实时调度算法的内部结构所针对的问题模式;

(2) Linux 实时调度算法对外呈现的接口所针对

收稿日期: 2014-03-21

修回日期: 2014-06-24

网络出版时间: 2014-12-27

基金项目: 国家核高基重大专项(2012ZX01039-004)

作者简介: 吴振亚(1989-), 男, 硕士研究生, 研究方向为 Linux 系统应用与网络; 李 航, 讲师, 研究方向为 Linux 内核可靠性、Linux 内核分析; 苏锐丹, 副教授, 研究方向为嵌入式 Linux 与网络安全。

网络出版地址: <http://www.cnki.net/kcms/detail/61.1450.TP.20141227.1348.044.html>

的问题模式。

对于(1),分为三部分进行论述。首先是实时调度算法组合框架,针对解决的问题模式是各种调度算法的组合模式;然后论述针对实时调度算法延迟问题的改进方案,这实际上论述的是各种实时调度算法普适性的改进方案;最后论述各种针对不同问题模式的调度算法。

此外,需要说明的是,由于 Linux 实时调度算法的工作非常庞杂,所以,在论述某一问题模式所采用的具体调度算法时,仅列出几项示例来说明,而非涵盖同类中所有的调度算法。

1 实时调度算法组合框架

实时调度算法组合框架针对的问题模式是:多种

表 1 slot 调度序列

		Slot #33	Slot #34	Slot #35	Slot #36	Slot #37	Slot #38	
CPU #1	...	PID 293	PID 293	PID 293	PID 293	PID 293	PID 37	...
CPU #2	...		PID 18	PID 37	PID 18		PID 18	...
CPU #3	...	PID 42	PID 42	PID 37			PID 37	...
CPU #4	...		PID 26	PID 37		PID 26	PID 37	...

基于时间槽的调度组合框架有许多变种,这些变种主要针对的问题是在多核系统中调度任务的迁移。主流的共有 3 种:基于 Partition、基于 Global 和 Task splitting^[4]。性能最好的是 Task splitting,它融合了 Partition 调度和 Global 调度的优点,避免了 Partition 调度和 Global 调度的缺点。既能够提高 CPU 的利用率,减少调度开销(对于多核体系架构,其中减少进程迁移代价的方法是利用多核体系架构中共享缓存和共享通信网络机制,使用缓存迁移),又能够确保可预测性(大部分的任务是静态地划分到指定核上来减小缓存迁移,剩下的任务允许以一种预先决定的规则迁移到预先选择好的 CPU 核的子集上,来增加任务集调度的可预测性)。因为在 partitioned 调度中,每个进程被固定分配到一个 core 上运行,每个 core 上使用一个调度器。而 Global 调度全局采用一个调度队列,允许进程在不同的 core 上迁移。Partition 调度虽然避免了进程迁移的开销,提高了可预测性,但会引发 bin-packing 这种问题,导致 CPU 的利用率低于全局调度,并且可能导致负载不均衡。而 Global 会引发全局调度队列的竞争问题,并且进程迁移会带来额外的开销。

2 实时调度算法延迟通用改进方案

由于所有的调度算法均存在一些共性机制,如加锁、中断等会影响实时调度算法的延时特性,所以可以针对这些共性机制,设计实现通用的减少实时调度算

实时非实时应用放在一起进行调度时,可调度性的预测问题。典型的有两种方式:

(1)基于优先级的调度算法组合方式。

这种调度算法组合框架下,按照任务的优先级进行调度,每类任务所使用的调度算法可以不同。但这种组合方式是最简单的一种,其最大的缺点在于:可调度性难以预测。因为在很多情况下,任务被高优先级抢占的时机很难甚至无法预测,从而导致一个任务究竟运行多长时间很难或无法预测。

(2)基于时间槽的调度组合框架。

该调度框架是将一个时间周期分为多个 slot,一个 CPU Core 上的一个 slot 只能运行一个进程,各个 CPU Core 上 slot 的开始和结束时间是一样的^[3],如表 1 所示。

法延迟的方案。目前,最著名的 Linux 实时调度算法延迟的改进工作是 RT patch,并已投入到企业实际应用中。其主要方法是:(1)使用 mutex 代替自旋锁;(2)中断线程化;(3)优先级继承;(4)指令替换。

2.1 使用 mutex 实现 spinlock

spinlock 是一个高效的共享资源同步机制,在 SMP (对称多处理器, Symmetric Multiple Processors) 的情况下,它用于保护共享资源,如全局的数据结构或一个只能独占的硬件资源。为了避免一个进程使用 spinlock 加锁后,未解锁之前,出现另一个高优先级进程抢占该进程,重入该 spinlock 所保护的关键区时,将导致死锁情况的发生(这种情况称为失效抢占);所以,持有 spinlock 锁期间 Linux 内核将使抢占失效(如通过关中断等方法),因此会严重地影响系统的实时性。比如:此时如果有一个根本不会加锁的高优先级的实时进程来到,那么,即便不会引发失效抢占,依然会导致 spinlock 无法使用。为此,实时补丁 PREEMPT_RT 使用 mutex 来实现 spinlock,它的意图是让 spinlock 可抢占。

但用 mutex 替换 spinlock 依然有可能产生失效抢占,为了避免失效抢占的发生,RT patch 采用等待队列来解决该问题:每个 spinlock 都有一个等待队列,该等待队列是按进程或线程的优先级排队的。如果一个进程或线程竞争的 spinlock 已经被另一个线程保持,它将自己挂在该 spinlock 的优先级化的等待队列上,然后发生调度把 CPU 让给别的进程或线程。等待队

列优先级化的目的是为了能够更好地改善实时性,因为优先级化后,每次当 spinlock 保持着释放锁时总是唤醒排在最前面的优先级最高的进程或线程,而唤醒的时间复杂度为 $O(1)$ 。

不过,使用 mutex 实现 spinlock 意味着:当进程调用 spin_lock() 后,进程有可能进入睡眠状态。如果进程在调用 spin_lock() 前,系统处于关中断状态或者禁止抢占的状态,此时,就不能调用 spin_lock()。因为,如果在进程中使用 spin_lock(),造成进程进入睡眠状态,将使进程陷入死锁状态。然而,内核在实际中有这样的应用场景,即在关中断后,需要调用 spin_lock()。PREEMPT_RT 对此的解决方案是:会延迟对 spin_lock() 的申请,直到抢占/中断又被重新开启为止。

2.2 中断线程化

PREEMPT_RT 补丁实现了上下半部 (Top-Half) 中断线程化。以往的 Linux 处理,仅使用上下半部 (top half, bottom half) 的机制,而不是将上下部中的任务完全当作一个线程进行处理。PREEMPT_RT 补丁则完全实现了下半部的中断线程化,即将下半部完全当作一个线程运行。

但在实际应用中,还存在着这样的情况,即:某些设备(注意:并不是所有设备)的上半部分也可以线程化,为了满足这种需求,RT patch 也提出了上半部线程化。为了区分出到底一个上半部是否需要线程化,RT patch 将该决定权交给驱动程序开发者决定。

2.3 优先级继承

RT patch 采用优先权继承的方案来解决优先级逆转问题。不过需要说明的是:解决优先级逆转的方案不止优先级继承这一种。

2.4 指令替换

在 x86 系统上的 MMX/SSE 指令集,这些指令集在内核中的操作都是在内核禁止抢占的情形下使用的。大部分的 MMX/SSE 指令的操作都是非常迅速的,但是仍有一些 MMX/SSE 指令操作的过程比较慢,RT patch 对此做出了优化,使用等价指令进行替换或者禁止使用这些操作比较慢的指令,这样可以提高中断响应的速度。

3 基于特定问题的实时调度算法

所有基于特定问题的实时调度算法的共性是满足任务的时限要求^[5-10],为此首先根据任务对时限要求的严格程度,将其分为硬实时调度算法^[11]和软实时调度算法。然后再分别列出这两种问题模式下的各个子问题模式。

在列举这些问题模式时,按照问题条件的严格程度,按照从严格到宽松的顺序一一列出。

3.1 硬实时调度算法

3.1.1 问题模式 1:上层任务的调度时机确切

对于这种问题模式,上层应用的调度时机确切,即操作系统开发者明确地知道上层究竟要运行哪些应用,并且这些应用的调度时机也是明确的。对于这种问题,最经典的算法是采用基于时间驱动 (Time-Driven 也称 Clock-Driven) 的进程调度算法。

基于时间驱动的进程调度算法本质上是一种设计时就确定下来的离线的静态调度方法^[12-13]。在系统的设计阶段,就明确各个任务的调度时机,对于各个任务的开始、切换以及结束时间等预先做出明确的安排和设计。这种调度算法适合于那些很小的嵌入式系统、自控系统、传感器等应用环境。这种调度算法的优点是任务的执行有很好的可预测性,但最大的缺点是缺乏灵活性,并且会出现有任务需要被执行而 CPU 却保持空闲的情况。典型的实现有 RED-Linux^[14]。

3.1.2 问题模式 2:上层任务的运行时机未知,仅知其时限要求

在这种问题模式下,由于任务的运行时机未知,所以,为了保证突然出现的任务能够及时完成,采用基于优先级的调度算法。各个基于优先级的调度算法的主要区别在于优先级的分配^[15-18]。典型的有 EDF (Earliest Deadline First)^[19-20] 和 RMS (Rate-Monotonic Scheduling)^[21]。

· EDF 调度算法。

最早期限优先 (EDF) 调度策略是动态优先级调度中最常见的实时调度策略。它以任务截止期限作为优先级,进程创建后被加到就绪队列中,该队列按照截止期限排序,首先运行截止时间最近的进程。在各种动态调度策略中,EDF 调度可得到最高的 CPU 利用率。

· 静态优先级调度算法 (RMS)。

这种调度算法给那些系统中得到运行的所有进程都静态地分配一个优先级。静态优先级的分配可以根据应用的属性来进行,比如任务的周期,用户优先级,或者其他预先确定的策略。RMS 调度算法是一种典型的静态优先级调度算法。该算法事先为每个进程分配一个与任务频率成正比的优先级,任务调度频率越高,优先级越高,调度程序总是调度优先级最高的就绪进程^[22-24]。RMS 策略为系统带来的额外负载非常得小,这对实时系统非常有利。RMS 算法假设任务有如下特点:

- (1) 在单个 CPU 中,所有进程都是周期性运行的;
- (2) 忽略上下文切换时间;
- (3) 进程间没有数据依赖关系,即各个进程相互独立,其完成和启动不依赖于其他进程的完成与启动;
- (4) 进程的执行时间是恒定的;

(5) 所有进程的最后期限都在它周期的结束点上;

(6) 优先级最高的就绪进程先被选择运行。

3.1.3 问题模式 3: 上层任务的运行时机未知, 运行时间难以预测

通常这种问题模式为实时调度带来的最大困扰是: 难以预测一个任务是否满足时限要求。运行时间难以预测的原因主要有如下几点:

1) 交互式任务, 任务的完成依赖于外界的输出, 那么, 任务的运行时间就依赖于:

(1) 外界输入完成的时间;

(2) 处理该输入所执行的特定路径。

实际中经常出现的情况是: 无法预知和不能精确预知。从而导致任务运行的时间难以预测。

2) 对于 Power-Aware 系统, 为了达到降低能耗的目的, 有时会降频运行任务, 这样会使执行时间延长。

所以, 此时实时调度算法所要解决的关键问题就是如何提高系统运行的可预测性。目前解决该问题的较好方式是采用资源预留策略^[25-28]。

其中, 一个较好地利用资源预留策略解决问题模式 3 而又能降低能耗的实时调度算法是 GRUB-PA (Greedy Reclamation of Unused Bandwidth—Power Aware, 根据文献[29], 采用 GRUB-PA 改进后的算法性能提高了百分之三十以上)。该算法能够用于硬实时和软实时任务共存的系统中, 而且这些任务可以是周期性任务、突发性任务或者非周期任务。其思想是: 调度器会为每个任务分配周期 P , 以及预留资源 Q , 即在周期 P 的时间段内, 该任务可以执行的时间长度为 Q 。如果任务执行的时间比预期的时间短, 可以回收这些未使用的时间, 并在这些未使用的时间片上降低 CPU 的频率去执行软实时或非实时的任务。

3.2 软实时调度算法

本节论述的软实时调度算法所针对的问题模式是: 软实时调度中多个任务均超时限^[30-31]。在此问题模式下, 最重要的就是保证各个人任务能够公平使用资源。其中较为典型的算法是: 基于比例共享的调度算法。虽然基于优先级的调度算法也是一种软实时调度算法, 但不适用于类似于实时多媒体会议系统这样的应用。此时, 基于比例共享式的资源调度算法 (SD 算法)^[32] 更为适合。

比例共享调度算法指基于 CPU 使用比例的共享式的调度算法, 其基本思想就是按照一定的权重 (比例) 对一组需要调度的任务进行调度, 让它们的执行时间与它们的权重完全成正比。不过, 比例共享调度算法没有定义任务的优先级, 所有的任务均按照预先设定的比例共享 CPU 资源。此时, 若当系统过载, 所

有的任务的执行都会变慢。所以为了保证系统中重要进程能够及早处理完毕, 一般采用动态调节进程权重的方法。

4 实时调度算法接口 API

如果仅有上述的章节中的实时调度算法, 而没有对应的为开发者提供的 API, 那么, 开发者开发实际的实时应用时就不够方便。目前, 实时调度算法用户接口 API 的改进所针对的问题模式是: 能够为用户提供丰富兼容的 API。其中, 最主流的 Linux 内核接口 API 的改进方案是: RTAI (Real-Time Application Interface)^[33] 和 Xenomai^[34]。RTAI 与 Xenomai 的区别在于: RTAI 虽然提供了丰富的 API, 但其接口种类只有 RTAI 一种, 没有提供兼容其他系统的 API; 而 Xenomai 不但提供了丰富的 API, 而且, 其接口种类繁多, 还提供了 Posix-RT, VMWORKS, VTRT 等多种兼容 API。此外, 由于实现上的差异, Xenomai 的中断处理延迟要比 RTAI 略高, 这是因为: 虽然 RTAI 和 Xenomai 同样基于 ADEOS, 但 RTAI 可以绕过 ADEOS 处理中断, 而 Xenomai 则不同。

5 结束语

通过上面的论述, 可以发现虽然 Linux 实时调度算法非常庞杂, 但是设计与实现是以问题为驱动的, 因而必然遵循一定的问题模式。文中识别出了其中的一些问题模式, 能够为企业在设计实现 Linux 实时调度算法时起到一定的指导作用。

参考文献:

- [1] Davis R I, Burns A. A survey of hard real-time scheduling for multiprocessor systems[J]. ACM Computing Surveys, 2011, 43 (4): 35-35.
- [2] Gantman A, Guo Peining, Lewis J, et al. Scheduling real-time tasks in distributed systems: a survey[R]. California: University of California, 1998.
- [3] Hall B. Slot scheduling: general-purpose multiprocessor scheduling for heterogeneous workloads[D]. Texas: The University of Texas at Austin, 2005.
- [4] Shekhar M, Sarkar A, Ramaprasad H, et al. Semi-partitioned hard-real-time scheduling under locked cache migration in multicore systems[C]//Proc of 24th Euromicro conference on real-time systems. Pisa, Italy: IEEE, 2012: 331-340.
- [5] Li Chuanpeng, Ding Chen, Shen Kai. Quantifying the cost of context switch [C]//Proc of ACM workshop experimental computer science. San Diego, CA: ACM, 2007.
- [6] Bril R J, Lukkien J J, Verhaegh W F J. Worst-case response time analysis of real-time tasks under fixed-priority schedu-

- ling with deferred preemption revisited [J]. Real-time Systems, 2009, 42(1-3): 63-119.
- [7] Leung J, Whitehead J. On the complexity of fixed-priority scheduling of periodic real-time tasks[J]. Performance Evaluation, 1982, 2(4): 237-250.
 - [8] Yao G, Buttazzo G, Bertogna M. Feasibility analysis under fixed priority scheduling with fixed preemption points [C]//Proc of 16th IEEE international conference on embedded real-time computing systems and applications. Macau SAR: IEEE, 2010: 71-80.
 - [9] Keskin U, Bril R J, Lukkien J J. Exact response-time analysis for fixed-priority preemption-threshold scheduling [C]//Proc of ETFA'10. Bilbao: IEEE, 2010: 1-4.
 - [10] Short M. Improved schedulability analysis of implicit deadline tasks under limited preemption EDF scheduling [C]//Proc of 16th ETFA'11. Toulouse, France: IEEE, 2011: 1-8.
 - [11] Liu C, Layland J. Scheduling algorithms for multiprogramming in a hard-real-time environment [J]. Journal of the ACM, 1973, 20(1): 46-61.
 - [12] Davis R I, Burns A, Bril R J, et al. Controllor Area Network (CAN) schedulability analysis: refuted, revisited and revised [J]. Real-time Systems, 2007, 35(3): 239-272.
 - [13] Bini E, Buttazzo G C. Schedulability analysis of periodic fixed priority systems [J]. IEEE Trans on Computers, 2004, 53(11): 1462-1473.
 - [14] Lin Kwei-Jay, Wang Yu-Chung. The design and implementation of real-time schedulers in RED-Linux [J]. Proceedings of the IEEE, 2003, 91(7): 1114-1130.
 - [15] Baruah S. The limited-preemption uniprocessor scheduling of sporadic task systems [C]//Proc of 17th Euromicro conference on real-time systems. [s. l.]: IEEE, 2005: 137-144.
 - [16] Burns A. Preemptive priority based scheduling: an appropriate engineering approach [M]. Upper Saddle River, NJ: Prentice Hall, 1995: 225-248.
 - [17] Saksena M, Wang Yun. Scalable real-time system design using preemption thresholds [C]//Proc of 21st IEEE real-time systems symposium. Orlando, FL: IEEE, 2000: 25-34.
 - [18] Altmeyer S, Davis R, Maiza C. Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems [C]//Proc of 32nd IEEE real-time systems symposium. Vienna, Austria: IEEE, 2011: 261-271.
 - [19] Faggioli D, Checconi F. An EDF scheduling class for the Linux kernel [C]//Proc of 11th real-time Linux workshop. Dresden, Germany: [s. n.], 2009.
 - [20] Bertogna M, Baruah S. Limited preemption EDF scheduling of sporadic task systems [J]. IEEE Trans on Industrial Information, 2010, 6(4): 579-591.
 - [21] Atlas A, Bestavros A. Design and implementation of statistical rate monotonic scheduling in KURT Linux [C]//Proc of 20th IEEE real-time systems symposium. Phoenix, AZ: IEEE, 1999: 272-276.
 - [22] Wang Y, Saksena M. Scheduling fixed-priority tasks with preemption threshold [C]//Proc of 6th IEEE international conference on real-time computing systems and applications. Hong Kong: IEEE, 1999: 328-335.
 - [23] Yao Gang, Buttazzo G, Bertogna M. Bounding the maximum length of non-preemptive regions under fixed priority scheduling [C]//Proc of 15th IEEE international conference on embedded and real-time computing systems and applications. Beijing: IEEE, 2009: 351-360.
 - [24] Bertogna M, Buttazzo G, Yao Gang. Improving feasibility of fixed priority tasks using non-preemptive regions [C]//Proc of 32nd IEEE real-time systems symposium. Vienna, Austria: IEEE, 2011: 251-260.
 - [25] Scordino C, Lipari G. Using resource reservation techniques for power-aware scheduling [C]//Proceedings of the 4th ACM international conference on embedded software. Pisa, Italy: ACM, 2004: 16-25.
 - [26] Regehr J. Scheduling tasks with mixed preemption relations for robustness to timing faults [C]//Proc of 23rd IEEE real-time systems symposium. Austin, TX: IEEE, 2002: 315-326.
 - [27] Facchinetti T, Vedova M D. Real-time modeling for direct load control in cyber-physical power systems [J]. IEEE Trans on Industrial Informatics, 2011, 7(4): 689-698.
 - [28] Bertogna M, Xhani O, Marinoni M, et al. Optimal selection of preemption points to minimize preemption overhead [C]//Proc of 23rd Euromicro conf on real-time systems. Porto, Portugal: IEEE, 2011: 217-227.
 - [29] Lipari G, Baruah S. Greedy reclamation of unused bandwidth constant-bandwidth servers [C]//Proceedings of the 12th Euromicro conference on real-time systems. Stockholm, Sweden: IEEE, 2000: 193-200.
 - [30] Yao Gang, Buttazzo G C, Bertogna M. Comparative evaluation of limited preemptive methods [C]//Proc of IEEE conference on emerging technologies and factory automation. Bilbao: IEEE, 2010: 1-8.
 - [31] Bini E, Buttazzo G C. Measuring the performance of schedulability tests [J]. Real-time Systems, 2005, 30(1-2): 129-154.
 - [32] Nieh J, Vaill C, Zhong Hua. Virtual-time round-robin; an O(1) proportional share scheduler [C]//Proceedings of the 2001 USENIX annual technical conference. CA, USA: USENIX Association, 2001: 245-259.
 - [33] Yaghmour K. The real-time application interface [C]//Proceedings of the Linux symposium. Boston, Massachusetts, USA: [s. n.], 2001: 245-260.
 - [34] Gerum P. Xenomai-implementing a RTOS emulation framework on GNU/Linux [EB/OL]. (2004) [2014-03-31]. <http://www.xenomai.org/documentation/xenomai-2.1/html/xenomai/>.

作者: [吴振亚](#), [李航](#), [苏锐丹](#), [WU Zhen-ya](#), [LI Hang](#), [SU Rui-dan](#)
作者单位: [吴振亚, 苏锐丹, WU Zhen-ya, SU Rui-dan \(西安电子科技大学 计算机学院, 陕西 西安, 710071\)](#), [李航, LI Hang \(西安电子科技大学 软件学院, 陕西 西安, 710071\)](#)
刊名: [计算机技术与发展](#) 
英文刊名: [Computer Technology and Development](#)
年, 卷(期): 2015 (2)

引用本文格式: [吴振亚](#). [李航](#). [苏锐丹](#). [WU Zhen-ya](#). [LI Hang](#). [SU Rui-dan](#) [Linux实时调度算法研究](#)[期刊论文]-[计算机技术与发展](#) 2015 (2)