软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(7):3422−3437 [doi: 10.13328/j.cnki.jos.006534]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于避让阻塞的优先级天花板协议*

陈 熙[1], 乔 磊[1], 杨孟飞[2], 刘洪标[1,3]

[1](北京控制工程研究所, 北京 100190)

[2](中国空间技术研究院, 北京 100094)

[3](西安电子科技大学 计算机科学与技术学院, 陕西 西安 710071)

通信作者: 乔磊, E-mail: fly2mars@163.com

摘 要: 为了提高空间飞行器计算机的 CPU 利用率, 新一代空间飞行器操作系统使用了一种同时包含固定时间点启动任务和偶发任务的混合调度算法. 其中固定时间点启动任务往往是安全攸关任务, 需要在固定时间点启动, 且执行期间不能被阻塞. 在固定时间点启动任务和偶发任务共存的条件下, 现有的实时锁协议无法保证固定时间点启动任务的阻塞时间为零, 因此在经典的优先级天花板协议的基础上, 提出基于避让思想的实时锁协议, 通过提前预判和设置虚拟启动点的方式, 确保偶发任务对共享资源的访问不会影响到固定时间点启动任务的执行. 同时暂时提升部分共享资源的访问优先级, 降低了任务抢占所带来的运行开销. 给出上述锁协议的最坏阻塞时间, 并通过可调度率实验分析其性能, 实验表明, 在临界区较短的情况下, 本协议可将因访问共享资源而导致的可调度性损失控制在 27% 以内.

关键词: 优先级天花板协议; 资源同步协议; 实时调度; 固定时间点启动; 避让阻塞

中图法分类号: TP306

中文引用格式: 陈熙, 乔磊, 杨孟飞, 刘洪标. 基于避让阻塞的优先级天花板协议. 软件学报, 2023, 34(7): 3422–3437. http://www.jos.org.cn/1000-9825/6534.htm

英文引用格式: Chen X, Qiao L, Yang MF, Liu HB. Priority Ceiling Protocol Based on Avoidance Blocking. Ruan Jian Xue Bao/Journal of Software, 2023, 34(7): 3422–3437 (in Chinese). http://www.jos.org.cn/1000-9825/6534.htm

Priority Ceiling Protocol Based on Avoidance Blocking

CHEN Xi[1], QIAO Lei[1], YANG Meng-Fei[2], LIU Hong-Biao[1,3]

[1](Beijing Institute of Control Engineering, Beijing 100190, China)

[2](China Academy of Space Technology, Beijing 100094, China)

[3](School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

Abstract: In order to improve the CPU utilization of spacecraft computers, the new generation of spacecraft operating system uses a hybrid scheduling algorithm that includes both fixed-point starting tasks and sporadic tasks. Among them, fixed-point starting tasks are often safety-critical tasks and need to be started at fixed points and cannot be blocked during execution. Under the condition that fixed-point starting tasks and sporadic tasks coexist, the existing real-time lock protocols cannot guarantee that the blocking time of fixed-point starting tasks is zero, so on the basis of the classic priority ceiling protocol, a real-time lock protocol based on the idea of avoidance blocking is proposed in this study to ensure that sporadic tasks' access to shared resources will not affect the execution of fixed-point starting tasks by judging in advance and setting virtual starting point. At the same time, by temporarily increasing the access priority of some resources, the cost caused by task preemption can be reduced. This study presents the worst blocking time of the above lock protocol and uses the schedulable rate experiments to analyze its performance. Experiments show that in the case of short critical sections, this protocol can control the schedulability loss caused by accessing shared resources to under 27%.

Key words: priority ceiling protocol; resource synchronization protocols; real-time scheduling; fixed-point starting; avoidance blocking

---

* 基金项目: 国家自然科学基金 (62032004, 61632005)
收稿时间: 2021-08-17; 修改时间: 2021-10-06; 采用时间: 2021-11-13; jos 在线出版时间: 2022-11-30
CNKI 网络首发时间: 2022-12-01

随着航天事业的发展, 航天任务逐渐复杂化、智能化, 空间飞行器计算机中的任务种类和数量也逐渐增加. 在空间飞行器计算机中, 数据采集、姿态控制、轨道计算等关键任务对实时性的要求高, 对时序的要求也极为严格, 需在事先规划好的固定时间点启动执行, 同时执行过程中不允许被抢占和阻塞[1]. 在这些任务之外, 还存在其他偶发任务, 这些任务到达时间随机, 也没有固定的启动时间, 只需保证在截止时间前完成执行即可. 此外, 任务之间往往需要共享内存、I/O 设备以及其他关键数据. 为了避免任务访问共享资源而引发的数据一致性问题, 需要使用锁协议来规范任务对共享资源的访问次序, 还应避免“死锁”和“优先级反转”现象. 同时, 由于固定时间点启动任务在执行过程中不能被阻塞, 因此在偶发任务和固定时间点启动任务共存的条件下, 锁协议还需保证固定时间点启动任务的最坏阻塞时间为零, 使得偶发任务对共享资源的访问不会影响到固定时间点启动任务的执行.

针对上述应用场景, 本文提出了基于避让阻塞的优级天花板协议 (avoidance blocking based priority ceiling protocol, APCP), 主要贡献如下.

(1) 协议保证了任务访问共享资源的正确性;

(2) 协议保证了固定时间点启动任务的最坏阻塞时间为零, 因此能够支持包含固定时间点启动任务和偶发任务的混合调度;

(3) 协议通过暂时提升部分共享资源的访问优先级, 降低了任务抢占所带来的运行开销.

本文首先介绍了研究背景, 而后第 1 节简要介绍相关工作. 第 2 节介绍任务模型. 第 3 节介绍本文所采用的优先级分配方法. 第 4 节介绍了本协议涉及的基本概念. 第 5 节详细描述了基于避让阻塞的优先级天花板协议. 第 6 节对上述协议的性质进行分析. 第 7 节展示了实验结果并对实验结果进行分析. 第 8 节总结了全文工作.

1 相关工作

实时锁协议最重要的设计原则是确保任务最坏阻塞时间可预测[2]. 最早进行实时锁研究的 Sha 等人, 在 1990 年针对 RM 调度算法提出了著名的优先级继承 (priority inheritance) 机制[3]: 当高优先级任务请求一个已经被低优先级任务占有的资源时, 将占有该资源的低优先级任务的优先级临时提升到与请求资源的高优先级任务一样的级别, 使得低优先级任务能更快地执行, 从而更快地释放资源. 当低优先级任务释放资源后, 再将其优先级恢复至初始值. 优先级继承机制防止了非临界区阻塞, 并将任务的最坏阻塞时间限制在 $\min(m,n)$ 个临界区中, 其中 m 为可以阻塞该任务的共享资源的数量, n 为优先级低于该任务的任务数. 虽然优先级继承机制解决了“未控优先级反转”的问题, 但无法避免“死锁现象”和“阻塞链”的产生. 为了避免上述两种现象, Sha 等人在优先级继承机制的基础上又提出了优先级天花板协议 (priority ceiling protocol, PCP)[3], 它为每个资源都设置了一个优先级天花板, 即需要访问该资源的最高优先级任务的优先级. 当任务请求某个资源时, 只有当该任务的优先级严格高于当前所有被锁住的资源的优先级天花板时, 该任务才可以锁住该资源, 否则将任务插入到该资源的等待队伍等待. 同时采用优先级继承机制将正在访问当前被锁住的优先级天花板最高的资源的任务的优先级暂时提升, 使其尽快释放所占有的资源. 优先级天花板协议可以将任务的最坏阻塞时间限制在一个临界区内, 具体来说, 任务最多被某一低优先级任务阻塞一次, 因此最大阻塞时间为可能阻塞该任务的低优先级任务临界区中, 临界区长度的最大值. 除此之外, PCP 协议还可避免“死锁”现象.

由于 PCP 协议对任务申请共享资源的限制过多, 可能会引起不必要的阻塞. Baker 等人对优先级天花板进行了改进, 提出了栈资源锁协议 (stack resource policy, SRP)[4], 主要用于 EDF 调度, 可同时支持互斥信号量和读/写锁. SRP 协议的特点是使用“提前阻塞”原则, 即任务在试图抢占其他运行中的任务时需判断该任务是否可能在执行后因访问共享资源而导致阻塞, 若会阻塞, 则调度器放弃此任务并继续寻找下一个合适的任务. 与 PCP 相比, SRP 这种提前阻塞的方法使得任务阻塞始于任务试图抢占的时刻, 而不是任务请求资源的时刻, 一旦新的任务通过了资源分配测试, 在后续的执行过程中就不会由于申请资源而阻塞. 这样做虽然降低了系统的并发性, 但也避免了不必要的上下文切换, 减小了系统开销, 同时也能避免未控优先级反转和死锁现象. 在任务最坏阻塞时间方面, PCP 和 SRP 具有相同的性能, 均被认为是单处理器最优实时锁协议, 同时可利用“资源分配测试”人为地限制任务

的并发性, 因此可支持共享任务堆栈, 这样可以极大节约堆栈空间. 然而 SRP 中的资源分配测试过程在调度器选择就绪队列中的最优任务时会消耗大量时间, 降低调度效率. 为此, 马运南等人引入了 K 路胜利树结构来改进就绪队列, 给出了新的任务搜索算法, 当任务规模较大时, 提高了调度效率[5].

Burns 等人将 PCP 协议扩展至混合关键度调度 (mixed criticality scheduling, MCS) 中, 提出了混合关键调度, 原始资源天花板协议 (mixed criticality scheduling, original priority ceiling protocol, MCS-OPCP), 但仅允许相同关键度的任务之间共享资源, 而不支持不同关键度的任务之间共享资源[6]. Lakshmanan 等人在零松弛调度 (zero slack scheduling, ZSS) 调度算法的基础上考虑了不同关键度任务间共享资源的情况, 提出了 PCIP (priority and criticality inheritance protocol) 和 PCCP (priority and criticality ceiling protocol), 但由于 ZSS 不适用于安全认证, 因此 PCIP 和 PCCP 也无法直接适用于可认证的 MCS 算法[7]. 浙江大学的赵庆玲等人分别基于 PCP 协议和 SRP 协议提出了针对自适应关键调度算法的资源同步协议 HLC-PCP (highest-locker criticality, priority-ceiling protocol)[8]和针对双重截止期调度算法的资源共享协议 MC-SRP (mixed-criticality stack resource protocol)[9].

在航天等安全关键领域中, 往往采用固定时间点启动调度, 这样做虽然能够保证系统的安全性和确定性, 但却降低了 CPU 利用率. 随着航天事业的不断发展, 传统的固定时间点启动调度无法满足日益增长的功能与性能需求, 因此系统需要在不执行固定时间点启动任务的空闲时间内执行偶发任务, 即将偶发任务与固定时间点启动任务进行混合调度. 但遗憾的是, 目前的锁协议仅适用于纯偶发任务系统, 无法支持偶发任务与固定时间点启动任务共存的混合调度系统. 针对上述问题, 本文基于避让阻塞的思想, 在经典的优先级天花板协议的基础上, 提出了一种适用于偶发任务和固定时间点启动任务混合调度的实时锁协议, 确保了偶发任务对共享资源的访问不会影响到固定时间点启动任务的执行, 保证了系统的正确性和实时性.

2 模型介绍

在航天器控制系统里, 存在数据采集、姿态控制、轨道计算等关键任务, 不同于一般的实时任务, 此类任务具有共同的周期 T_c, 称为系统控制周期 (system control period), 它们之间后者依赖前者, 需要严格地按照不同的固定时间点启动执行, 由于截止时间等于最差执行时间, 因此执行过程中不可被其他任务抢占和阻塞, 这类任务我们称为固定时间点启动任务 (fixed-point starting task), 简称“固定点任务”. 而其他实时任务具有各自的周期, 在满足最小间隔的条件下到达时刻任意, 只需保证在其截止时间前能够完成即可, 并且任务间也无固定执行顺序, 这类任务我们称为偶发任务 (sporadic task). 本文考虑的系统中同时存在上述两种任务, 在固定时间段内执行固定点任务, 在固定点任务不占用 CPU 的空闲时间内执行偶发任务, 以提高空间飞行器计算机的 CPU 利用率[1].

2.1 系统模型

本文考虑单核实时系统, 该系统具有 m ($m \geqslant 1$) 个固定点任务组成的任务集合 $\Pi = \{\gamma_1, \ldots, \gamma_m\}$, 其中 γ_h ($1 \leqslant h \leqslant m$) 表示第 h 个固定点任务; 具有 n ($n \geqslant 1$) 个偶发任务组成的任务集合 $\Gamma = \{\tau_1, \ldots, \tau_n\}$, 其中 τ_i ($1 \leqslant i \leqslant n$) 表示第 i 个偶发任务; 以及 q ($q \geqslant 1$) 个共享资源组成的共享资源集合 $S = \left\{\rho^1, \ldots, \rho^q\right\}$, 其中 ρ^r ($1 \leqslant r \leqslant q$) 表示第 r 个共享资源. 本文仅考虑互斥型共享资源, 即同一时刻, 某一共享资源仅能被一个任务访问.

2.2 任务模型

在实时系统中, 每个任务均周而复始地执行, 固定点任务的一次执行被称为一个固定点作业, 偶发任务的一次执行被称为一个偶发作业. 所有固定点任务的周期都是相同的, 均为控制周期 T_c, 但各自的初始到达偏移 (arrival offset) 互不相同, 每个任务需要严格地按照其到达偏移在相应的时间点启动. 一个固定点任务用 $\gamma_h=(A_h, M_h)$ 来描述, 其中 A_h 表示第 h 个固定点任务的初始到达偏移, M_h 表示第 h 个固定点任务的最坏执行时间, 将第 h 个固定点任务的第 w 次执行表示为 λ_h^w, 其到达时间记为 $r\left(\lambda_h^w\right)$. 一个偶发任务用 $\tau_i = (T_i, C_i, D_i)$ 来描述, 其中 T_i 为周期, 表示任务 τ_i 的任意相邻作业间的最小到达时间间隔, C_i 表示任务 τ_i 的最坏执行时间, D_i 表示任务 τ_i 的相对截止时间, 即 τ_i 的每个作业均需在 D_i 时间内完成, 将第 i 个偶发任务的第 j 次执行表示为 J_i^j, 其到达时间记为 $r\left(J_i^j\right)$. 若 $D_i=T_i$, 则 τ_i 称为隐式截止时间 (implicit deadline) 任务, 若 $D_i \leqslant T_i$, 则称 τ_i 为约束截止时间 (constrained deadline) 任务, 若

D_i 与 T_i 无明确关系, 则称 τ_i 为非约束截止时间 (arbitrary deadline) 任务. 本文考虑的系统中, 所有偶发任务均为约束截止时间任务, 以上描述的任务模型执行示例如图 1 所示.

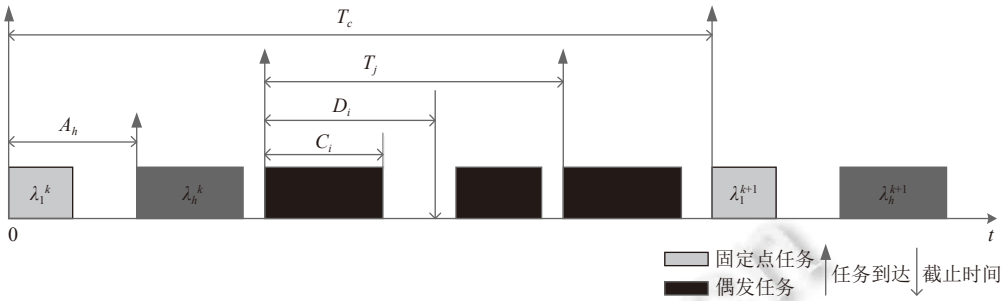

图 1 固定点任务和偶发任务共存的任务模型示例

3 优先级分配策略

优先级调度策略[10,11]可以分为固定优先级调度和动态优先级调度, 固定优先级调度又分为固定优先级非抢占式调度 (FPNS), 固定优先级抢占调度 (FPPS) 和固定优先级抢占延迟调度 (FPDS). 本文采用固定优先级抢占式调度. 由于固定点任务在执行过程中不允许被其他任务抢占, 因此固定点任务应比偶发任务拥有更高的优先级. 同时又由于固定点任务拥有相同的周期和不同的到达偏移, 因此需按照固定点任务的初始到达偏移的大小为其分配优先级, 即初始到达偏移量越小的固定点任务, 其优先级越高, 反之亦然. 由于系统中所有偶发任务的周期都不相等, 因此可以采用 RM 调度算法[12]来对偶发任务分配优先级, 即周期越小的任务, 其优先级越高, 反之亦然. 令 $pr(\tau_i)$、$pr(\gamma_h)$ 分别表示偶发任务 τ_i 和固定点任务 γ_h 的基本优先级, $pr(\tau_i)$ 越大, 表示偶发任务 τ_i 的优先级越高, $pr(\gamma_h)$ 越大, 表示固定点任务 γ_h 的优先级越高. 为简化叙述, 本文假设任务按基本优先级降序排列, 即任务编号越小其基本优先级越高. 例如, $pr(\tau_1)$ 为基本优先级最高的偶发任务, $pr(\tau_n)$ 为基本优先级最低的偶发任务, τ_i 的基本优先级高于 τ_j 当且仅当 $i<j$; $pr(\gamma_1)$ 为基本优先级最高的固定点任务, $pr(\gamma_m)$ 为基本优先级最低的固定点任务, γ_h 的基本优先级高于 γ_w 当且仅当 $h<w$. 其中固定点任务的优先级高于所有偶发任务的优先级, 即 $pr(\gamma_m)>pr(\tau_1)$. 根据共享资源协议规则, 系统运行时偶发任务的有效优先级可能高于其基本优先级, 因此, 偶发任务 τ_i 在 t 时刻的有效优先级表示为 $pr_i(t)$. 系统运行时, 操作系统根据任务的运行时有效优先级来进行任务调度. 由于 APCP 的需要, 还需定义一个高于所有偶发任务但低于所有固定点任务的“关键优先级” (critical priority), 记为 prm, 满足 $pr(\gamma_m)>prm>pr(\tau_1)$.

例 1: 如后文图 2 所示, 系统中存在 4 个任务, 2 个固定点任务: γ_1=(0, 4), γ_2=(7, 3), 2 个偶发任务: $\tau_1=(10,2,8)$, $\tau_2=(30,5,25)$. 由于 γ_1 和 γ_2 为固定点任务, τ_1 和 τ_2 为偶发任务, 根据优先级设置策略, γ_1, γ_2 优先级都应高于 τ_1, τ_2 的优先级. 又由于 $A_1<A_2$, 因此 $pr(\gamma_1)>pr(\gamma_2)$; 由于 $T_1<T_2$, 因此 $pr(\tau_1)>pr(\tau_2)$. 综上, 此系统中的优先级设置结果应为 $pr(\gamma_1)>pr(\gamma_2)>pr(\tau_1)>pr(\tau_2)$.

4 相关定义

定义 1. 关键资源与非关键资源.

按照资源是否会被固定点任务访问到, 将资源分为关键资源 (crucial resource) 和非关键资源 (non-crucial resource). 既会被偶发任务访问, 又会被固定点任务访问到的共享资源为关键资源; 仅会被偶发任务访问到的共享资源为非关键资源.

将偶发任务 τ_i 需要访问的共享资源的集合记为 S_i, $S_i\subseteq S$; 将偶发任务 τ_i 需要访问的关键资源的集合记为 S_i^c, $S_i^c\subseteq S_i$; 将固定点任务 γ_h 需要访问的共享资源的集合记为 C_h, $C_h\subseteq S$.

定义 2. 长资源与短资源.

按照访问资源需要的时间长短, 又将资源分为长资源 (long resource) 和短资源 (short resource), 这里的访问时间长短完全由用户来指定, 一般来说, 共享内存等资源可分类为短资源, 而 I/O 设备等资源可分类为长资源.

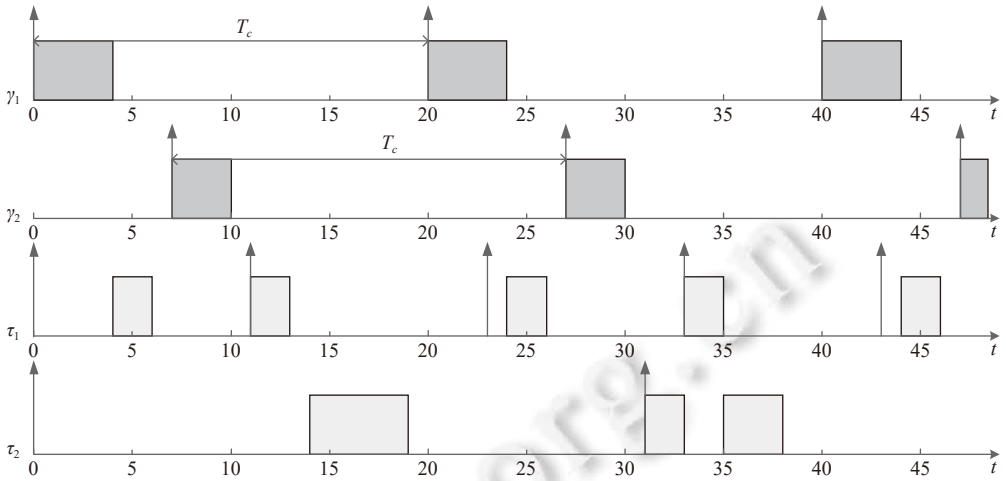

图 2 优先级分配策略举例

定义 3. 资源访问时间.

将偶发作业 J_i^j 对共享资源 ρ^k 的第 n 个请求记为 $R_{i,j}^k(n)$, 请求时刻记为 $t\left(R_{i,j}^k(n)\right)$, 访问 ρ^k 的最大时长记为 $\left|R_{i,j}^k(n)\right|$. 若 J_i^j 正在执行访问 ρ^k 的程序, 则称 J_i^j 正处于 ρ^k 的临界区中.

定义 4. 资源紧邻作业.

在 t 时刻, 所有即将需要访问资源 ρ^k 的固定点作业中, 最先到达的作业称为资源 ρ^k 在 t 时刻的资源紧邻作业 (next job in resource), 记为 $NxtRJob^k(t)$.

定义 5. 空闲时间差.

时间差一般定义为时间轴上两个时间点的差值, 但由于本文考虑的系统中存在固定点任务, 因此将时间轴上两个时间点的差值再减去这期间固定点任务的执行时间总和定义为空闲时间差 (free time difference). t_2 时刻与 t_1 时刻的空闲时间差用 $\langle t_2-t_1\rangle$ 来表示, 其意义为 t_2 时刻与 t_1 时刻之间, 除去固定点任务所需的执行时间后, 可以用来执行偶发任务的空闲时间的时长. 如图 3 所示, $t=25$ 时刻与 $t=5$ 时刻的时间差为 25 – 5 = 20, 而 $t=25$ 时刻与 $t=5$ 时刻的空闲时间差为 $\langle 25-5\rangle$=25 – 5 – (M_1+M_2) = 25 – 5 – (6 + 3) = 11. 类似地, t 时刻之前 Δt 个空闲时间单位所对应的时刻用 $[t-\Delta t]$ 来表示. 如图 3 所示, $t=25$ 时刻之前 8 个时间单位所对应的时刻为 t = 25 – 8 = 17, 而 $t=25$ 时刻之前 8 个空闲时间单位所对应的时刻为 [25 – 8] = 14.

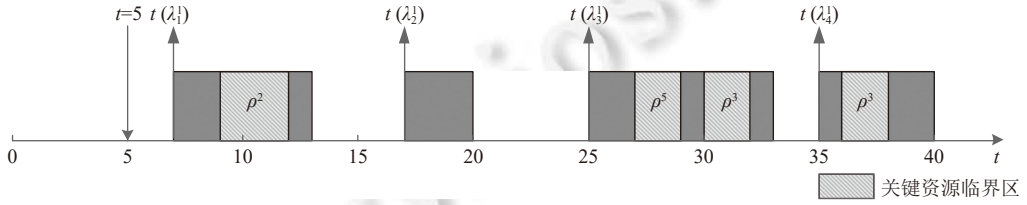

图 3 空闲时间差示例

定义 6. 虚拟启动点.

若在 t 时刻, 关键资源 ρ^k 的资源紧邻作业为 λ_h^w, 则定义 λ_h^w 在 t 时刻的虚拟启动点 (virtual starting point) 为 $p_h^w(t)=[r\left(\lambda_h^w\right)-RT^k(t)]$, 其中 $RT^k(t)$ 表示在 t 时刻距离 ρ^k 被释放的时长. 还以图 3 为例, 在 $t=5$ 时刻, 假设此时距某偶发作业释放 ρ^3 还有 8 个时间单位, 因此在 $t=5$ 时刻, $NxtRJob^3(5)=\lambda_3^1$, $p_3^1(5)=[r\left(\lambda_3^1\right)-RT^3(5)]$=[25–8]=14.

从理论上讲, 在访问 ρ^k 期间, $RT^k(t)$ 的值在每个时间单位都需要更新. 由于处于 ρ^k 临界区的作业可能会因高优先任务的抢占而耽搁对 ρ^k 的访问, 导致在 $r\left(\lambda_h^w\right)$ 前无法释放 ρ^k. 而设置变量 $RT^k(t)$ 的目的就是为了记录当处于 ρ^k 临界区的作业被抢占时, ρ^k 此时被访问的进度, 以便求出该作业的最晚恢复时间 (即 λ_h^w 的虚拟启动点), 使得该作业及时释放关键资源, 以保证 λ_h^w 能够正常执行. 因此实际上, $RT^k(t)$ 只有在正处于 ρ^k 的临界区的偶发作业被其他任务抢占的时刻或退出临界区的时刻才有意义, 而其他时刻的值没有意义, 因此仅需在正处于 ρ^k 的临界区的偶发作业被其他任务抢占的时刻或退出临界区的时刻更新 $RT^k(t)$ 即可, 没有必要在每个时间单位都更新 $RT^k(t)$ 的值. 而在虚拟启动点的计算公式中, 只有 $RT^k(t)$ 是动态变化的, $r\left(\lambda_h^w\right)$ 是固定值, 因此 λ_h^w 的虚拟启动点也是随着 ρ^k 的访问进度而不断更新, 由于 $RT^k(t)$ 的值只在正处于 ρ^k 的临界区的偶发作业被其他任务抢占的时刻或退出临界区时发生变化, 因此仅需在上述时间点更新 $p_h^w(t)$ 即可.

需要注意的是, 资源紧邻作业是针对关键资源而言的, 在某一时刻, 某个关键资源的资源紧邻作业就是确定的, 在该资源紧邻作业进入该关键资源临界区之前都不会发生改变. 而虚拟启动点是针对于即将到来的固定点作业的, 当某一固定点作业需要访问的某个资源事先被某个偶发作业占有时, 该固定点作业的虚拟启动点会随着该资源的访问进度而不断更新, 直到该资源被释放.

定义 7. 松弛时间.

若在 t 时刻, 偶发作业 J_i^j 第 n 次请求关键资源 ρ^k, 则将 ρ^k 的资源紧邻作业 λ_h^w 的到达时间 $r\left(\lambda_h^w\right)$ 与 $t\left(R_{i,j}^k(n)\right)$ 的空闲时间差定义为请求 $R_{i,j}^k(n)$ 的松弛时间 (laxity time), 记为 $LxTime\left(R_{i,j}^k(n)\right)=\left\langle r\left(\lambda_h^w\right)-t\left(R_{i,j}^k(n)\right)\right\rangle$. 松弛时间是针对某一请求而言的, 是固定值, 将作为判断能否对资源 ρ^k 上锁的条件.

例 2: 如图 4(a) 所示, 当 $t=5$ 时, 偶发作业 J_4^2 到达并执行并在 $t=7$ 时刻第 1 次请求关键资源 ρ^2, 由于 λ_3^2 是第 1 个需要访问 ρ^2 的固定点任务, 因此此时 ρ^2 的资源紧邻作业为 λ_3^2, 即 $NxtRJob^2(7)=\lambda_3^2$. 根据松弛时间的定义, 此时请求 $R_{4,2}^2(1)$ 的松弛时间为 $LxTime\left(R_{4,2}^2(1)\right)=\left\langle r\left(\lambda_3^2\right)-t\left(R_{4,2}^2(1)\right)\right\rangle=\langle 40-7\rangle=33$.

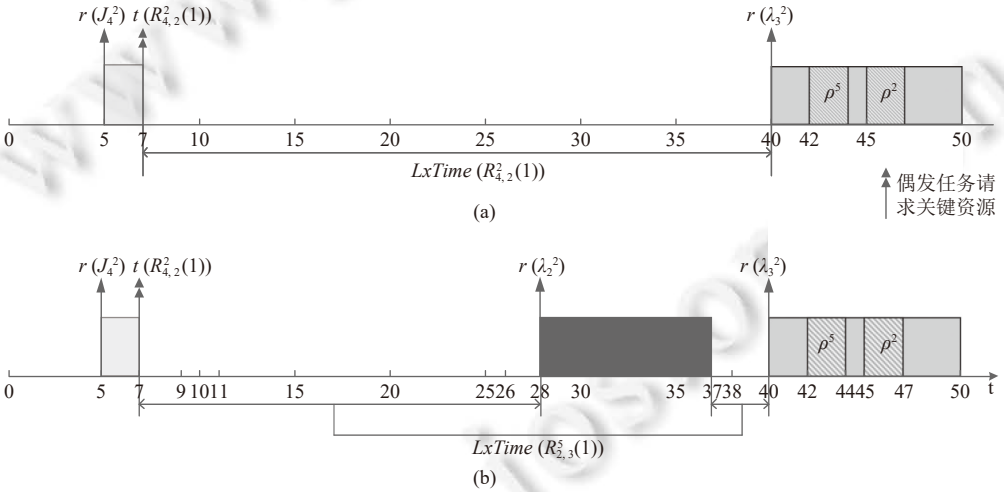

图 4 松弛时间示例

例 3: 考虑偶发任务和其相关的固定时间点任务间存在其他固定时间点任务的情况, 如图 4(b) 所示, λ_3^2 与 J_4^2 之间存在固定点任务 λ_2^2=(28,9). 根据松弛时间的定义, 此时请求 $R_{4,2}^2(1)$ 的松弛时间为 $LxTime\left(R_{4,2}^2(1)\right)=\left\langle r\left(\lambda_3^2\right)-t\left(R_{4,2}^2(1)\right)\right\rangle=\langle 40-7\rangle=40-7-9=24$.

5 基于避让阻塞的优先级天花板协议

定义资源 ρ 的天花板优先级 $PC(\rho)$ 为所有使用该资源的任务中, 基础优先级最高的任务的优先级. 将某一

时刻除自身外所有被锁住的资源中天花板优先级最高的资源记为 $\rho^*(t)$, 并将资源 $\rho^*(t)$ 的天花板优先级定义为此时的系统天花板优先级 $\psi(t)$, 同时将占有资源 $\rho^*(t)$ 的偶发作业记为 $J^*(t)$, 并将 $J^*(t)$ 的有效优先级记为 $pr_*(t)$.

APCP 协议的主体设计思路依然沿用 PCP 协议的设计思路, 包括优先级继承机制、优先级天花板机制等, 但在偶发任务申请共享资源时进行了进一步的限制: 除了要满足 PCP 协议中规定的申请者优先级要高于系统优先级以外, 还应保证偶发任务能在其资源紧邻作业到达之前可以释放关键资源, 这样才可以锁住关键资源. 在锁住关键资源后, APCP 协议还通过设置虚拟启动点和暂时提高短资源访问优先级的方式来确保偶发任务能够及时释放关键资源并减少系统开销, 最后还明确了当任务发生避让阻塞后的唤醒时机, 即当相应的资源紧邻作业执行结束后, 阻塞在该作业上的偶发作业应当全部被唤醒.

5.1 基于避让阻塞的优先级天花板协议定义

(1) 偶发任务申请共享资源: 在时刻 t, 偶发作业 J_i^j 第 n 次请求共享资源 ρ^k. 若 J_i^j 的有效优先级不高于此时的系统天花板优先级, 则 J_i^j 阻塞, 并将 J_i^j 按优先级顺序插入 ρ^k 的等待队列中, 若 J_i^j 的有效优先级高于此时任务 $J^*(t)$ 的有效优先级 $pr_*(t)$, 则 $J^*(t)$ 应继承此时 J_i^j 的有效优先级, 即将 $pr_*(t)$ 设置为 $pr_i(t)$. 若 J_i^j 的有效优先级高于此时的系统天花板优先级, 则需判断 ρ^k 是否为关键资源. 若 ρ^k 为非关键资源, 则 J_i^j 锁住 ρ^k; 若 ρ^k 为关键资源, 则需判断请求 $R_{i,j}^k(n)$ 的访问时长是否超过其松弛时间. 若请求 $R_{i,j}^k(n)$ 的访问时长不超过其松弛时间, 即 $\left|R_{i,j}^k(n)\right| \leqslant LxTime\left(R_{i,j}^k(n)\right)$, 则 J_i^j 锁住资源 ρ^k; 否则 J_i^j 阻塞, 并将 J_i^j 插入至 $NxtRJob^c(t)$ 的等待队列队尾. 当 $NxtRJob^c(t)$ 执行结束后, 将其等待队列中的作业全部唤醒.

(2) 偶发任务执行临界区: 当 J_i^j 锁住 ρ^k 后, 若 ρ^k 为非关键资源, 则 J_i^j 访问 ρ^k; 若 ρ^k 为关键短资源, 则 J_i^j 需将其有效优先级提升至关键优先级并访问资源 ρ^k; 若 ρ^k 为关键长资源, 则 J_i^j 开始访问资源 ρ^k, 并且在 J_i^j 被抢占或退出 ρ^k 的临界区的时刻更新 ρ^k 的资源紧邻作业的虚拟启动点.

(3) 偶发任务释放共享资源: 若占有资源 ρ^k 的偶发作业 J_i^j 结束对 ρ^k 的访问并释放 ρ^k, 若 J_i^j 进行过优先级提升操作, 则需将 J_i^j 的优先级恢复至占有资源 ρ^k 前的优先级, 并将 ρ^k 等待队列中的首个任务唤醒.

(4) 为了降低锁协议的复杂度, 规定偶发作业访问关键资源时不可嵌套访问任何共享资源, 即 J_i^j 在占有关键资源 ρ^k 后, 不可再次访问其他任何类型的共享资源 (包括关键资源与非关键资源), 直至 J_i^j 释放 ρ^k.

5.2 基于避让阻塞的优先级天花板协议资源申请算法

第 5.1 节 (1) 中介绍的偶发任务申请共享资源的方法是 APCP 协议的核心所在, 也是最能体现出避让思想的部分, 将该方法以伪代码的形式给出, 如算法 1.

算法 1. APCP 资源申请方法.

输入: 申请共享资源的偶发作业 J_i^j, 申请的共享资源 ρ^k;

输出: 申请结果, TRUE 或者 FALSE (TRUE 表示资源申请成功, FALSE 表示资源申请失败).

```
BOOL PendResource(J_i^j, ρ^k)
1. IF (pr_i(t) > ψ(t))
2.   IF (ρ^k is critical resource)
3.     IF (|R_{i,j}^k(n)| ⩽ LxTime(R_{i,j}^k(n)))
4.       IF (ρ^k is short resource)
5.         pr_i(t) ← prm
6.         J_i^j owns ρ^k
7.         RETURN TRUE
8.       ELSE
```

9. startUpdatePoint(J_i^j, $NxtRJob^k(t)$) //开始在J_i^j被抢占或退出ρ^k临界区的时刻, 更新ρ^k的资源紧邻作业的虚拟启动点.
10. J_i^j owns ρ^k
11. RETURN TRUE
12. ELSE
13. blockByFixedTask(J_i^j, $NxtRJob^k(t)$) // J_i^j阻塞, 并插入到ρ^k的资源紧邻作业的等待队列队尾, 待该固定点作业结束, 唤醒队列中所有作业.
14. RETURN FALSE
15. ELSE
16. J_i^j owns ρ^k
17. RETURN TRUE
18. ELSE
19. blockByResource(J_i^j, ρ^k) // J_i^j阻塞, 并将其插入到ρ^k的等待队列队尾, 待ρ^k被释放, 唤醒队列中的队首作业.
20. RETURN FALSE

5.3 APCP 协议中的阻塞

原始的 PCP 协议具有 3 种阻塞: 直接阻塞、间接阻塞和天花板阻塞, 而 APCP 协议由于考虑了固定点任务, 因此原始 PCP 协议的基础上引入了瞬时阻塞、紧要阻塞以及避让阻塞. 上述几种阻塞具体的定义如下.

● 原始 PCP 协议中的阻塞

阻塞 1: 直接阻塞 (direct blocking)

当高优先级偶发作业J_H申请的资源正在被低优先级偶发作业J_L访问, 则称J_H被J_L直接阻塞.

阻塞 2: 间接阻塞 (push-through blocking)

若正处于某个资源临界区的低优先级偶发作业J_L因继承了高优先级偶发作业J_H的优先级而抢占中等优先级偶发任务J, 则称J被J_L间接阻塞.

阻塞 3: 天花板阻塞 (ceiling blocking)

当高优先级偶发作业J_H申请某个资源时, 若J_H低于此时的系统优先级天花板, 则称J_H发生天花板阻塞.

● 关键资源引发的阻塞

阻塞 4: 瞬时阻塞 (instant blocking, AIB)

在低优先级偶发作业J_L正在访问关键短资源的期间, 由于J_L的有效优先级暂时提升至关键优先级而抢占高优先级偶发作业J_H的现象称作J_H被J_L瞬时阻塞.

阻塞 5: 紧急阻塞 (urgent blocking, AUB)

在时刻t, 低优先级偶发作业J_L处于关键资源ρ^c的临界区时, 且$NxtRJob^c(t)=\lambda$. 当λ的虚拟启动点到达后, J_L将以关键优先级继续访问该临界区, 导致高优先级偶发作业J_H无法执行, 将上述现象称作J_H被J_L紧急阻塞.

阻塞 6: 避让阻塞 (avoidance blocking, AAB)

偶发作业J申请关键资源ρ^c时, 当J的优先级高于此时的系统天花板优先级, 但该请求的松弛时间低于其请求时长, 导致J无法占有关键资源ρ^c的现象, 称为J发生了避让阻塞.

将直接阻塞、间接阻塞和天花板阻塞统称为“PCP 阻塞”; 并且由于一个偶发作业仅会发生 PCP 阻塞、瞬时阻塞和紧要阻塞中的一种阻塞 (详见第 6 节定理 3), 因此将以上 3 类阻塞统称为“ACP 阻塞”. 综上, APCP 协议中一共存在两大类阻塞——ACP 阻塞和避让阻塞. APCP 中各类阻塞关系如图 5 所示.

例 4: 如图 6 所示的系统中存在 4 个固定点任务: $\gamma_1=(0,5)$, $\gamma_2=(15,2)$, $\gamma_3=(18,5)$, $\gamma_4=(26,8)$, 且控制周期$T_c=41$; 存在 3 个偶发任务: $\tau_1=(35,4,28)$, $\tau_2=(45,6,40)$, $\tau_3=(60,12,50)$.

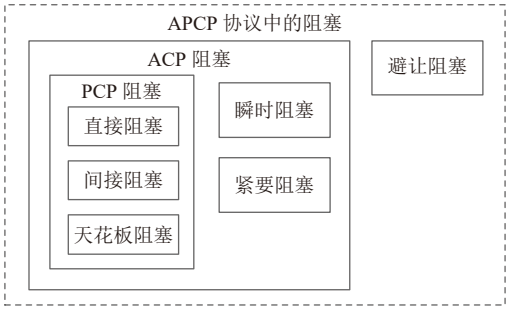

图 5 APCP 协议中的阻塞关系图

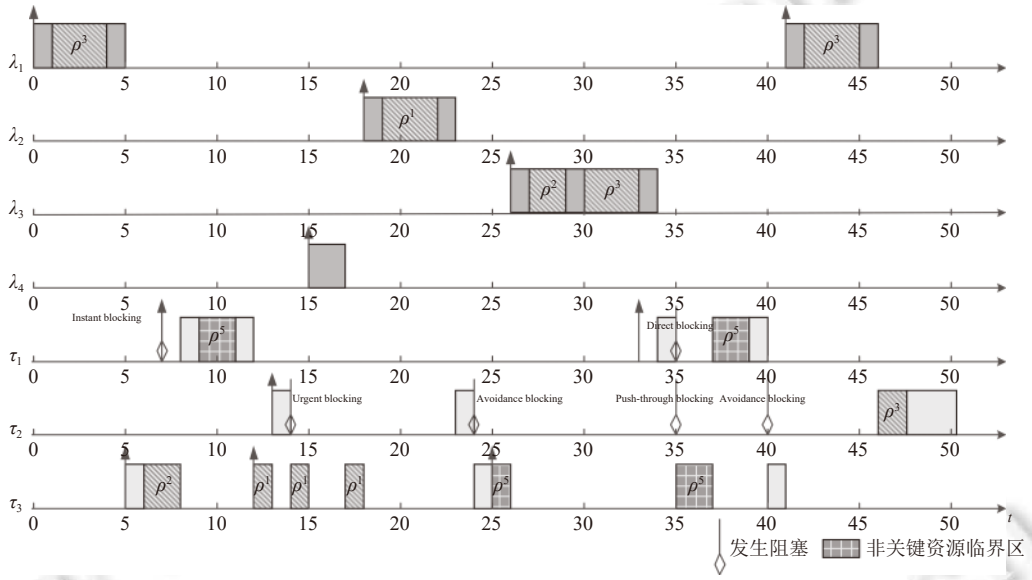

图 6 APCP 协议举例

在 $t=0$ 时, 固定点作业 λ_1^1 到达并执行. $t=5$ 时, λ_1^1 执行完成同时偶发作业 J_3^1 到达并执行. $t=6$ 时, J_3^1 第 1 次请求关键短资源 ρ^2, 显然此时 ρ^2 的资源紧邻作业为 λ_4^1, 因此此时请求 $R_{3,1}^2(1)$ 的松弛时间 $LxTime\left(R_{3,1}^2(1)\right)=\left\langle r\left(\lambda_4^1\right)-t\left(R_{3,1}^2(1)\right)\right\rangle=\langle 26-6\rangle=13$, 而 J_3^1 对 ρ^2 的访问时长 $\left|R_{3,1}^2(1)\right|$ 为 2, 由于 $\left|R_{3,1}^2(1)\right|\leqslant LxTime\left(R_{3,1}^2(1)\right)$, 因此 J_3^1 可以锁住资源 ρ^2 并访问. 在此期间高优先级偶发作业 J_1^1 到达, 由于 J_3^1 以关键优先级访问关键短资源 ρ^2, 因此 J_1^1 无法抢占 J_3^1, 此时 J_1^1 发生的阻塞为“瞬时阻塞”. $t=8$ 时, J_3^1 结束对 ρ^2 的访问并将优先级恢复至初始优先级, 而后 J_1^1 立刻抢占 J_3^1 并执行. $t=12$ 时, J_1^1 执行结束, 而后 J_3^1 继续执行并第 1 次申请关键长资源 ρ^1, 此时 ρ^1 的资源紧邻作业为 λ_3^1, 因此 $LxTime\left(R_{3,1}^1(1)\right)=\left\langle r\left(\lambda_3^1\right)-t\left(R_{3,1}^1(1)\right)\right\rangle=\langle 18-12\rangle=4$, 由于 $\left|R_{3,1}^1(1)\right|=3\leqslant LxTime\left(R_{3,1}^1(1)\right)$, 因此 J_3^1 成功申请 ρ^1 并访问. $t=13$ 时, J_2^1 到达并抢占 J_3^1, 由于此时距 ρ^1 释放还有 2 个时间单位, 因此更新 λ_3^1 的虚拟启动点为 $[r\left(\lambda_3^1\right)-RT^1(13)=14]$. J_2^1 执行 1 个时间单位后, 由于到达了 λ_3^1 的虚拟启动点, 因此 J_3^1 以关键优先级抢占 J_2^1 并继续访问 ρ^1, 此时 J_2^1 发生的阻塞为紧要阻塞, $t=14$ 时 λ_2^1 执行, λ_2^1 执行结束后, J_3^1 继续访问 ρ^1, 并于 $t=18$ 释放 ρ^1 并恢复优先级, 而后 λ_3^1 开始执行. $t=23$ 时 λ_3^1 结束, J_2^1 继续执行并于 $t=24$ 时第 1 次申请关键长资源 ρ^3, 此时 $NxtRJob^3(24)=\lambda_4^1$, $LxTime\left(R_{2,1}^3(1)\right)=\left\langle r\left(\lambda_4^1\right)-t\left(R_{2,1}^3(1)\right)\right\rangle=\langle 26-24\rangle=2$, 由于 $\left|R_{2,1}^3(1)\right|=3>LxTime\left(R_{2,1}^3(1)\right)$, 因此 J_2^1 无法申请 ρ^3, 此时 J_2^1 发生的阻塞为避让阻塞. 随后 J_3^1 继续执行并于 $t=25$ 访问 ρ^5, $t=26$ 时 λ_4^1 到达并执行, $t=34$ 时, λ_4^1 结束, J_1^1 再次执行并于 $t=35$ 时被 J_3^1 阻塞, 此时 J_1^1 发生的阻塞为直接阻塞, 因此 J_3^1 继承了 J_1^1 的优先级而继续访问 ρ^5, 此时 J_2^1 发生的阻塞为间接阻塞, $t=37$ 时 J_3^1 释放 ρ^5 并恢复优先级, 而后 J_1^1 继续执行. $t=40$ 时 J_1^1 执

行结束, 而后 J_2^1 第 2 次尝试申请关键资源ρ^3, 此时$NxtRJob^3(40)=\lambda_1^2$, $LxTime\left(R_{2,1}^3(2)\right)=\left\langle r\left(\lambda_1^2\right)-t\left(R_{2,1}^3(2)\right)\right\rangle=\langle 41-40\rangle=1$, 由于$\left|R_{2,1}^3(2)\right|=3>LxTime\left(R_{2,1}^3(2)\right)$, 因此$J_2^1$依然无法申请$\rho^3$, 因此由$J_3^1$继续执行. $t=41$时, J_3^1执行结束同时λ_1^2到达并执行. $t=46$时, λ_1^2执行结束, J_2^1第 3 次尝试申请资源 ρ^3,此时 $NxtRJob^3(40)=\lambda_4^2$, $LxTime\left(R_{2,1}^3(3)\right)=\left\langle r\left(\lambda_4^2\right)-t\left(R_{2,1}^3(3)\right)\right\rangle=\langle 67-46\rangle=14$, 由于$\left|R_{2,1}^3(3)\right|=3\leqslant LxTime\left(R_{2,1}^3(3)\right)$, 因此$J_2^1$可以申请并访问$\rho^3$. $t=50$时, J_2^1执行结束.

6 基于避让阻塞的优先级天花板协议的性质

由于 PCP 协议可以避免死锁现象, 而 APCP 协议在此基础上进一步增强了任务访问共享资源的限制, 没有造成对资源额外的申请, 因此 APCP 协议也可以避免死锁现象. 同时, 由 APCP 协议可知, 若偶发作业能成功申请某个关键资源, 则此偶发作业就一定能在与其资源相关的固定点作业到达之前释放该资源, 因此可确保固定点作业的阻塞时间为零, 使得偶发作业对共享资源的访问不会影响到固定点作业的执行. 下面将对 APCP 协议中任务的最坏阻塞时间进行分析.

将能对任务τ_i形成阻塞的请求的集合记为β_i. 将β_i中相互嵌套的请求合并, 仅保留外层的请求同时忽略内层的请求而形成的集合记为β_i^*, 即若$R_i^m\in\beta_i, R_i^n\in\beta_i$, 同时$R_i^m\subset R_i^n$, 则$R_i^n\in\beta_i^*, R_i^m\notin\beta_i^*$. 将一个控制周期内, 需要访问关键资源$\rho^c$的固定点任务的个数记为$m(\rho^c)$, 将需要访问到$\rho^c$的固定点任务称为$\rho^c$涉及的固定点任务. 令$n_i=\max\left\{m(\rho),\forall\rho\in S_i^c\right\}$, 即$n_i$表示在$\tau_i$需要访问的关键资源中, 涉及到的固定点任务数的最大值.

引理 1. 当有关键资源被占有后, 系统中就不会再有新的资源被偶发作业占有, 直到该关键资源被释放.

证明: 根据优先级天花板的定义, 当偶发作业J占有关键资源ρ^c后, 由于ρ^c是固定点任务需要访问的资源, 因此ρ^c的优先级天花板应不低于$pr(\gamma_m)$. 根据 APCP 协议中对于偶发任务申请共享资源的规定, 如果后续有偶发作业申请共享资源, 需要该作业的优先级高于目前被占有的所有资源的优先级天花板的最高值. 由于偶发作业的优先级低于ρ^c的优先级天花板, 因此J后续的偶发任务无法占有任何资源. 而J本身在没有释放ρ^c的情况下, 由于 APCP 不允许在关键资源之间嵌套访问任务资源, 因此J在没有释放ρ^c之前也无法占有其他任何资源. 综上, 当有关键资源被占有后, 系统中就不会再有新的资源被偶发作业占有, 直到该关键资源被释放.

定理 1. 同一时刻, 系统中仅有一个关键资源被偶发作业占有.

证明: 由引理 1, 当偶发作业J占有关键资源ρ^c后, 不会再有任何资源被偶发作业占有, 因此, 就不会再有任何关键资源被偶发作业占有. 在偶发作业J占有关键资源ρ^c之前, 由引理 1 同理可得, 既然J能占有关键资源ρ^c, 说明在此之前没有偶发作业占有任何关键资源, 否则J无法占有关键资源ρ^c.

引理 2. 在 APCP 协议下, 若偶发作业能成功申请某个关键资源, 则此偶发作业就一定能在其资源紧邻作业到达之前释放该资源.

证明: APCP 协议中具有两个机制以保证申请到关键资源的偶发作业能够在该关键资源的资源紧邻作业到达之前释放该资源. 首先第 1 个机制, 在 APCP 协议中, 偶发作业申请关键资源时要判断其请求资源的访问时长是否超过其松弛时间 (由定义可知, 松弛时间表示在其资源紧邻作业到达之前还剩多少时间可以用来访问关键资源), 若访问时长超过其松弛时间, 则说明偶发作业无法在资源紧邻作业到达之前完成对该关键资源的访问, 即该偶发作业无法在资源紧邻作业到达之前释放该关键资源, 按照 APCP 协议, 该偶发作业将被阻塞; 若该偶发作业申请关键资源的访问时长不超过其松弛时间, 则说明在资源紧邻作业到达之前的这段时间足够偶发作业完成对该关键资源的访问, 因此该偶发作业具有可以在资源紧邻作业到达之前就释放关键资源的可能性, 但单此一个条件还无法确保偶发作业一定能在资源紧邻作业到达之前就释放关键资源, 比如在偶发作业访问该关键资源时, 如果在此期间有高优先级作业到达并且抢占访问关键资源的偶发作业, 则偶发作业就会耽搁对关键资源的访问, 可能使得偶发作业在资源紧邻作业到来之前无法释放该资源, 为了避免这种情况, APCP 协议为固定点作业设置了虚拟启动点, 固定点作业在某一时刻的虚拟启动点应设置为该作业的到达时间与该时刻距离关键资源被释放的剩余时间(即此时还需多少时间单位该偶发作业才能释放关键资源), 也就是说, 在虚拟启动点到来之际就必须继续访问偶发作业的关键资源了, 否则在固定点作业到来前无法结束对关键资源的访问. 按照 APCP 的定义, 当虚拟启动点到

来时, 访问关键资源的偶发作业的优先级会被提升到高于一切偶发任务的关键优先级, 这段时间是不会再被高优先级偶发作业抢占的, 而根据虚拟启动点的定义可知, 在虚拟启动点继续执行临界区的偶发作业刚好可以执行完关键资源临界区, 因此偶发作业一定是可以在相应资源紧邻作业到达之前释放该关键资源的.

定理 2. 在 APCP 协议下, 固定点任务的最坏阻塞时间为 0.

证明: 由引理 2 可知, 偶发作业一定能在其资源紧邻作业到达之前释放该资源, 因此在任何一个访问关键资源的固定点任务到达时, 该任务需要访问的关键资源一定可用, 因此固定点任务的最坏阻塞时间一定为 0.

引理 3. 在 APCP 协议下, 偶发作业 J_L 能对 J 形成 ACP 阻塞的前提条件是: 当 J 就绪时, 有正处于临界区的低优先级作业 J_L.

证明: ACP 阻塞是由于低优先级任务访问关键资源而导致高优先级任务无法执行的现象, 因此低优先级任务必须获得执行的机会. 当高优先级任务 J 处于就绪态后, J_L 只能通过优先级继承或者瞬时加速、紧急抢占的方式来获得执行的机会, 但上述 3 种方式均要求当 J 就绪时 J_L 必须处于临界区当中.

引理 4. 偶发作业 J_L 仅会对 J 形成一次 ACP 阻塞.

证明: 由于 J_L 会对 J 形成 ACP 阻塞, 由引理 3, J 就绪时, J_L 处于能够阻塞 J 的最长临界区之一. 当 J_L 一旦离开上述临界区, J_L 就无法再次获得执行的机会, 由引理 2, J_L 也就无法再对 J 形成 ACP 阻塞.

引理 5. 偶发作业最多发生一次 ACP 阻塞.

证明: 由引理 4, 一个高优先级作业 J 仅会被一个低优先级作业 J_L 阻塞一次, 下面只需证明当被 J_L 阻塞后, J 不会再被其他低优先级作业阻塞. 假设被 J_L 阻塞后, J 又被 J_L' 阻塞, 根据引理 2, 当 J 就绪时, J_L 和 J_L' 也应在临界区中, 不妨设 J_L 先于 J_L' 进入临界区, 同时设在 t_1 时刻 J_L 进入访问 ρ 的临界区. 由于 J_L 能阻塞 J, 说明 J 的优先级低于 ρ 的优先级天花板. 随后 J_L' 进入临界区, 根据 APCP, 说明 J_L' 的优先级高于 ρ 的优先级天花板, 这与我们的假设 J_L' 的优先级低于 J 的优先级相矛盾, 因此 J_L 阻塞 J 以后, 不会再有其他低优先级任务再次阻塞 J.

定理 3. 不访问关键资源的偶发作业最多会被低优先级作业阻塞一次, 且该阻塞必为 ACP 阻塞.

证明: APCP 协议中的阻塞共分为两类: ACP 阻塞和避让阻塞. 避让阻塞只有在偶发作业申请关键资源时才可能发生, 显然不访问关键资源的偶发作业是不会发生避让阻塞的, 又根据引理 5, 因此偶发作业仅会发生 ACP 阻塞, 且最多发生一次.

定理 4. 偶发作业 J_i 最多发生 $\lceil D_i/T_c \rceil \times n_i$ 次避让阻塞以及 $\lceil D_i/T_c \rceil \times n_i + 1$ 次 PCP 阻塞.

证明: J_i 在每次申请关键资源 ρ^c 时都发生避让阻塞, 由于避让阻塞都是在 ρ^c 涉及的固定时间点任务之前发生的, 因此 D_i 期间 J_i 最多发生的避让阻塞次数与这期间 ρ^c 涉及的固定时间点任务数量有关. 下面将按照 D_i 与 T_c 的关系来讨论 J_i 在 D_i 期间发生的最大阻塞次数. 当 $D_i = T_c$ 时, 在 D_i 期间 J_i 最多发生的避让阻塞和 PCP 阻塞的最大次数如图 7 中所示, 其中仅画出控制周期内 ρ^c 所涉及的 4 个固定时间点任务, 即 $m(\rho^c) = 4$. 显然, 当 $D_i = T_c$ 时, D_i 期间内 J_i 最多会发生 $m(\rho^c)$ 次避让阻塞和 $m(\rho^c)+1$ 次 PCP 阻塞, 即如图中 d_3 所示 (d_1, d_2 和 d_4 都代表其他阻塞次数低于 d_3 的情况); 当 $D_i < T_c$ 时, 显然, D_i 期间依然在 d_3 的情况下能发生最大阻塞次数, 即 $m(\rho^c)$ 次避让阻塞和 $m(\rho^c)+1$ 次 PCP 阻塞.

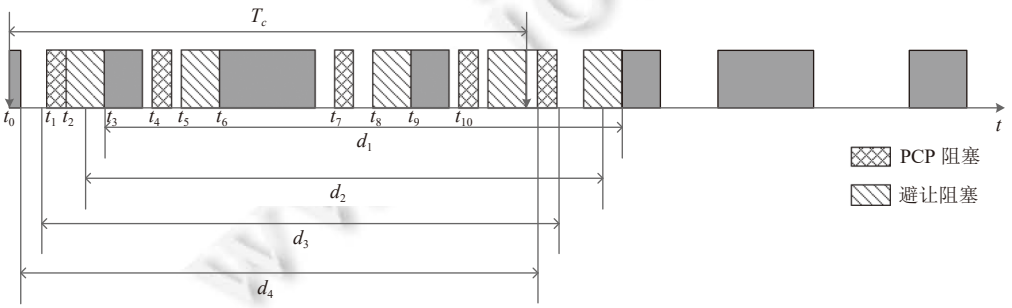

图 7 访问关键资源的偶发任务最坏阻塞时间分析

同理, 当$D_i = kT_c (k > 1 \wedge k \in N)$时, J_i最多会发生$(D_i/T_c)\times m(\rho^c)$次避让阻塞和$(D_i/T_c)\times m(\rho^c)+1$次 PCP 阻塞; 当$D_i = (k-1)T_c + m(k > 1 \wedge k \in N, 0 < m < T_c)$时, 显然, D_i期间最多发生$(D_i/T_c)\times m(\rho^c)$次避让阻塞和$(D_i/T_c)\times m(\rho^c)+1$次 PCP 阻塞. 而$J_i$不止会访问$\rho^c$一个关键资源, 在访问完$\rho^c$后, 还有可能访问其他的关键资源$\rho^d$, $\rho^d \in S_i^c$, 在访问ρ^d时, 一个控制周期中可能会发生更多的避让阻塞, 因此J_i在D_i的最大阻塞次数应以S_i^c中涉及到的固定时间点任务数最大的资源来计算, 因此J_i在D_i期间最多发生$\lceil D_i/T_c \rceil \times n_i$次避让阻塞和$\lceil D_i/T_c \rceil \times n_i + 1$次 PCP 阻塞.

定理 5. 在 APCP 协议下, 偶发任务不会发生“优先级反转”.

证明: 由共享资源导致的优先级反转是指当低优先级任务阻塞高优先级任务期间, 该低优先级任务又被后续到达的中等优先级任务抢占, 造成高优先级任务除了要等待锁住共享资源的低优先级任务, 还要被迫等待中等优先级任务执行, 也就是说, 高优先级任务的阻塞时间可能与其他任务的非临界区长度有关, 若在高优先级被阻塞期间, 有无数个中等优先级任务到达并抢占低优先级任务, 就会导致高优先级任务的阻塞时间无限长, 最终导致高优先级任务错过了截止期, 这种情况对实时系统的危害性是很大的, 因此在锁协议的设计上就应该杜绝这种“优先级反转”的现象, 即保证偶发任务的最坏阻塞时间是可控的, 而非无限长的, 此外在准入控制的可调度性分析也需要偶发任务具有一个确定的最坏阻塞时间. 而本文中通过第六章的定理 3 和定理 4 说明 APCP 协议可以使得偶发作业的最坏阻塞时间是固定值, 而非不确定的、无限长的, 而且都仅与偶发任务的临界区有关, 因此 APCP 可以避免“优先级反转”现象.

本协议的设计初衷是为了实现一种适用于偶发任务与固定点任务共存系统的实时锁协议, 也就是说, 协议首先要保证偶发任务对共享资源的访问不会影响到固定点任务的执行. 即固定点任务的最坏阻塞时间为 0. 而在定理 2 中已经明确地证明了 APCP 可以保证固定点任务的最坏阻塞时间为 0; 同时还要确保锁协议可以避免“死锁”和“优先级反转”的问题, 由于 APCP 协议是基于 PCP 协议来设计的, 而 PCP 协议是可以避免“死锁”的, 同时 APCP 协议又是在 PCP 协议的基础上进一步加强对偶发任务申请共享资源的限制的, 因此 APCP 协议是可以避免“死锁”现象的, 而对于“优先级反转”现象来说, 定理 5 中已经明确地证明了 APCP 可以避免“优先级反转”现象. 综上, APCP 协议可以满足协议设计的两个初衷, 即保证固定点任务最坏阻塞时间为 0, 以及可以避免“死锁”和“优先级反转”现象.

7 实验及结果分析

7.1 实验方案

本文通过 4 组仿真实验, 分别对比了 APCP 协议在不同临界区长度、任务数量、偶发任务占比以及短资源占比下任务集合的可调度率. 本实验的可调度率分析方法采用响应时间分析 (response time analysis, RTA), RTA 是最常用的可调度性分析方法[13,14], 能精确地给出任务集的可调度性判定, 得到了学术界广泛的研究和应用.

实验参数生成方案介绍如下.

● 任务集数量: 产生随机任务集, 任务集 CPU 利用率 U 应处于 (0, 1] 区间内, 每个任务的 CPU 利用率采用 UUnifast 算法[15]生成, 任务集 CPU 利用率 U 生成的步长为 0.04. 对于每一个任务集 CPU 利用率 U, 都生成 100 个任务集, 这 100 个任务集中可调度的任务集的占比将作为该 CPU 利用率 U 下的可调度率.

● 任务周期: 任务的周期长度服从均匀分布, 取值范围为 [1, 9999], 并将第一次随机产生的周期确定为控制周期T_c, 固定点任务的周期都相同且为控制周期, 对于偶发任务, 每次都随机产生, 可各不相同.

● 任务最坏执行时间: 偶发任务τ_i的最坏执行时间为其 CPU 利用率与其周期的乘积, 即$C_i = u_i \cdot T_i$, 若C_i小于 0, 则重新产生该任务周期值, 直到最差执行时间C_i大于 0 为止; 固定点任务的最坏执行时间为其 CPU 利用率与控制周期的乘积.

● 相对截止时间: 对于偶发任务τ_i而言, 在成功生成周期和最坏执行时间后, 相对截止时间的生成应服从均匀分布, 且取值范围为 $[T_i - (T_i - C_i)\times 0.8, T_i]$, 如图 8 所示.

● 任务到达偏移: 对于固定点任务而言, 到达偏移应服从均匀分布, 且取值范围为 [0, T_c).

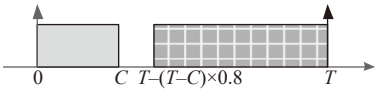

图 8 相对截止时间生成示意图

● 共享资源使用情况: 共享资源数为 4 个, 各任务按照概率 0.25 随机确定其使用的共享资源, 当某一任务确定要访问资源 ρ 后, 该任务访问 ρ 的次数在 [1,3] 区间内随机取值.

实验 1. 以任务集利用率为自变量, 对比 APCP 协议在不同临界区长度下的可调度率. 任务集数、任务周期、任务最坏执行时间、相对截止时间、任务到达偏移和共享资源使用情况的参数保持如上, 其余各参数如下.

● 任务数: 每个任务集的任务总数为 $n = 30$.

● 偶发任务占比: 2/3, 即任务集中包含 20 个偶发任务, 10 个固定点任务.

● 短资源占比: 0.5, 即共享资源中包含 2 个长资源, 2 个短资源.

● 在实验 1 中, 主要评测偶发任务的长、短资源临界区长度对任务集可调度率的影响, 这里我们评测了 4 种临界区长度组合下的可调度性. 具体来说, 我们在每个 CPU 利用率下, 为每种临界区长度组合都生成一个任务集 (即每个 CPU 利用率下总计生成 400 个任务集), 具体生成方法如下: 第 1 类任务集中偶发任务不访问共享资源, 即长资源与短资源的临界区长度均为 0; 第 2 类任务集中偶发任务访问短资源的最大时间和长资源的最大时间分别在区间 [1, 2] ms 和区间 [2, 5] ms 内随机取值, 且服从均匀分布; 第 3 类任务集中偶发任务访问短资源的最大时间和长资源的最大时间分别在区间 [2, 5] ms 和区间 [5, 20] ms 内随机取值, 且服从均匀分布; 第 4 类任务集中偶发任务访问短资源的最大时间和长资源的最大时间分别在区间 [5, 20] ms 和区间 [20, 40] ms 内随机取值, 且服从均匀分布. 根据上述任务参数产生方法, 任务最坏执行时间与其对共享资源访问的最大时间由相互独立的过程产生, 当任务的最坏执行时间小于其访问的共享资源最大时间总和时, 将重新生成上述参数, 直至任务的最坏执行时间大于其访问的共享资源最大时间总和.

实验 2. 以任务集利用率为自变量, 对比 APCP 协议在不同任务数下的可调度率. 任务集数、任务周期、任务最坏执行时间、相对截止时间、任务到达偏移和共享资源使用情况的参数保持如上, 其余各参数如下.

● 偶发任务占比: 2/3.

● 短资源占比: 0.5, 即共享资源中包含 2 个长资源, 2 个短资源.

● 临界区长度: 偶发任务访问短资源的最大时间和长资源的最大时间分别在区间 [2, 5] ms 和区间 [5, 20] ms 内随机取值, 且服从均匀分布.

● 在实验 2 中, 主要评测任务集合的任务数对任务集可调度率的影响, 这里我们评测了任务总数为 20, 30 和 40 下的任务集的可调度性.

实验 3. 以任务集利用率为自变量, 对比 APCP 协议在不同偶发任务占比下的可调度率. 任务集数、任务周期、任务最坏执行时间、相对截止时间、任务到达偏移和共享资源使用情况的参数保持如上, 其余各参数如下.

● 任务集任务数: 任务集任务总数为 $n = 30$.

● 短资源占比: 0. 5, 即共享资源中包含 2 个长资源, 2 个短资源.

● 临界区长度: 偶发任务访问短资源的最大时间和长资源的最大时间分别在区间 [2, 5] ms 和区间 [5, 20] ms 内随机取值, 且服从均匀分布.

● 在实验 3 中, 主要评测偶发任务占比对任务集可调度率的影响, 这里我们评测了偶发任务占比为 0.067, 0.500, 0.667 和 0.933 下的任务集的可调度性.

实验 4. 以任务集利用率为自变量, 对比 APCP 协议在不同短资源占比下的可调度率. 任务集数、任务周期、任务最坏执行时间、相对截止时间、任务到达偏移和共享资源使用情况的参数保持如上, 其余各参数如下.

● 任务集任务数: 任务集任务总数为 $n = 30$.

● 偶发任务占比: 2/3.

● 临界区长度: 偶发任务访问短资源的最大时间和长资源的最大时间分别在区间 [2, 5] ms 和区间 [5, 20] ms

内随机取值, 且服从均匀分布.

● 在实验 4 中, 主要评测短资源占比对任务集可调度率的影响, 这里我们评测了偶发任务占比为 0.5 和 1 下的任务集的可调度性.

7.2 实验结果及分析

图 9(a) 是实验 1 的结果, 显示了不同临界区长度组合下任务集的 CPU 利用率从 0.25 逐渐增加到 1 时任务集在 APCP 协议下的可调度性. 可以看出, CPU 利用率越高, 可调度率越低; 临界区越长, 可调度率越低. 任务集的 CPU 利用率越高, 由于任务访问共享资源而导致的可调度性的损失就越明显, 例如, 当临界区长度组合为 [2, 5] ms 和 [5, 20] ms 时, 随着 CPU 利用率从 0.28 增长至 0.52, 可调度率从 0.32 降低至 0.02. 另外, 随着临界区长度增加, 可调度性的损失也越明显, 例如, 当 CPU 利用率为 0.35 时, 随着临界区长度增加, 可调度率从 0.42 下降至 0.08. 由于现实应用中临界区长度通常很短[16], 因此临界区长度组合 [1, 2] ms 和 [2, 5] ms 代表着最实际的情况, 在这种情况下, 当任务集 CPU 总利用率不超过 0.6 时, 与临界区长度为 0 相比, 可调度率下降不超过 27%, 说明在 APCP 协议下, 由访问共享资源而导致的可调度性损失通常在 27% 以内.

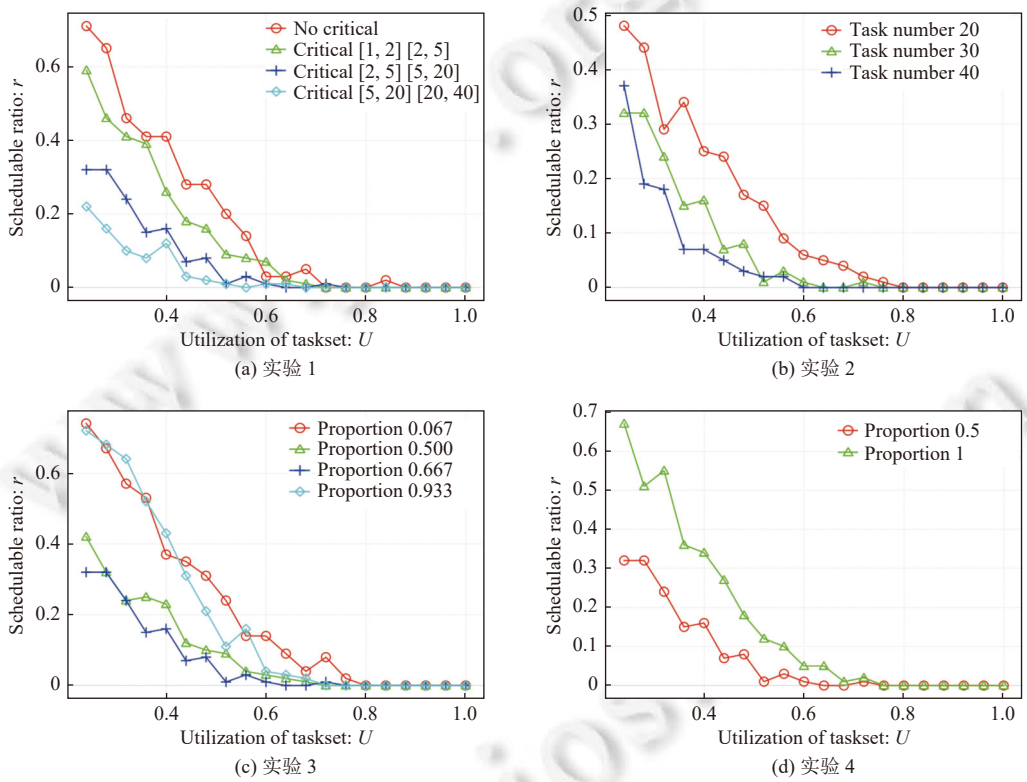

图 9 短资源占比不同对任务集可调度率的影响

图 9(b) 是实验 2 的结果, 显示了不同任务数下任务集的 CPU 利用率从 0.25 逐渐增加到 1 时任务集在 APCP 协议下的可调度性. 可以看出, CPU 利用率越高, 可调度率越低; 任务数越大, 可调度率越低, 任务数的增加导致高优先级任务间抢占量增加, 低优先级任务容易错过截止期, 使得可调度率下降.

图 9(c) 是实验 3 的结果, 显示了不同偶发任务占比下任务集的 CPU 利用率从 0.25 逐渐增加到 1 时任务集在 APCP 协议下的可调度性. 可以看出, CPU 利用率越高, 可调度率越低; 而随着偶发任务占比升高, 可调度率却不是单调递增的, 如图中 4 条代表不同偶发任务占比的曲线所示, 当 CPU 利用率低于 0.6 时, 偶发任务占比为 0.067 的可调度率较高, 当偶发任务占比提高至 0.5 时, 可调度率降低, 当偶发任务占比提高至 0.667 时, 可调度率继续降

低, 当偶发任务占比提高至 0.933 时, 可调度率不增反降, 并且提高至与偶发任务占比为 0.067 时相当的水平. 可见可调度率并不是随偶发任务占比提高而单调降低的, 当偶发任务占比较低时, 虽然固定点任务数量较多, 但偶发任务数量较少, 可以有比较充足的时间来执行, 因此可调度率较高; 随着偶发任务占比较提高, 导致偶发任务之间的抢占、阻塞增加, 使得任务集的可调度率下降; 而后又随着偶发任务占比进一步提高, 虽然偶发任务数量增加, 但固定点任务数量降低了, 这就可以留给偶发任务更多的执行时间, 因此利用率不降反增.

图 9(d) 是实验 4 的结果, 显示了不同短资源占比下任务集的 CPU 利用率从 0.25 逐渐增加到 1 时任务集在 APCP 协议下的可调度性. 可以看出, CPU 利用率越高, 可调度率越低; 短资源占比越高, 可调度率越高, 短资源占比越高, 短资源数量越多, 使得偶发任务的阻塞时间降低, 使得任务集可调度率上升.

从整体上看, 我们研究的是各参数 (临界区长度、任务数、偶发任务占比和短资源占比) 在 CPU 利用率递增情况下, 对任务集可调度率的影响. 因此, 尽管这些参数不尽相同, 但是 CPU 负载是不断增大的, 因此导致任务集可调度率下降趋势相似. 然而, 在同一个 CPU 利用率下, 各参数不同就会导致不同的可调度率, 该结果对锁协议的进一步研究具有重要的指导意义.

8 总 结

本文针对同时包含固定点任务和偶发任务的混合调度算法, 根据任务的实时性要求对任务进行了优先级分配, 并且提出了一种基于避让思想的优先级天花板协议, 通过提前判断的方式进一步限制了偶发任务对关键资源的申请, 将无法在相应固定点任务到达之前完成关键资源访问的偶发任务进行了阻塞, 同时为固定点任务设置虚拟启动点, 保证了偶发任务对共享资源的访问不会影响到固定点任务的执行; 并且将关键资源按照访问时间长短进行分类, 通过暂时提升短资源的访问优先级的方法降低了任务切换所导致的开销. 本文通过严格的理论分析证明了上述锁协议可以避免“死锁”和“优先级反转”现象, 并给出了最坏阻塞时间的理论值, 最后通过可调度率的仿真实验对其性能进行了测试和分析, 实验结果显示, 本协议可将由任务访问共享资源而导致的可调度率损失控制在 27% 以内. 而本协议的最坏阻塞时间分析结果还不够精确, 可能会导致可调度率偏低, 因此后续会对其最坏阻塞时间的分析工作做进一步的研究. 同时, 随着航天任务不断复杂化, 使用多核处理器来提升算力是大势所趋, 因此下一步将会考虑将本协议拓展至多核处理器.

References:

[1] Qiao L, Gong J, Yang MF, Yang H, Peng F, Xu J, Tan YL, Wu YF, Liu ZH. A method for scheduling spacecraft periodically mixed random tasks: China, 201610814726.8. 2017-02-22 (in Chinese).

[2] Yang ML. Research on real-time scheduling algorithms with shared resources [Ph.D. Thesis]. Chengdu: University of Electronic Science and Technology of China, 2016 (in Chinese with English abstract).

[3] Sha L, Rajkumar R, Lehoczky JP. Priority inheritance protocols: An approach to real-time synchronization. IEEE Trans. on Computers, 1990, 39(9): 1175–1185. [doi: 10.1109/12.57058]

[4] Baker TP. Stack-based scheduling of realtime processes. Real-time Systems, 1991, 3(1): 67–99. [doi: 10.1007/BF00365393]

[5] Ma YN, Chen XL. Real-time scheduling technique under SRP protocol. Computer Systems & Applications, 2016, 25(2): 1–8 (in Chinese with English abstract).

[6] Burns A. The application of the original priority ceiling protocol to mixed criticality systems. 2013. https://www.cs.york.ac.uk/rts/static/papers/R:Burns:2013f.pdf

[7] Lakshmanan K, de Niz D, Rajkumar R. Mixed-criticality task synchronization in zero-slack scheduling. In: Proc. of the 17th IEEE Real-time and Embedded Technology and Applications Symp. Chicago: IEEE, 2011. 47–56. [doi: 10.1109/RTAS.2011.13]

[8] Zhao QL, Gu ZH, Yao M, Zeng HB. HLC-PCP: A resource synchronization protocol for certifiable mixed criticality scheduling. Journal of Systems Architecture, 2016, 66–67: 84–99. [doi: 10.1016/j.sysarc.2016.01.008]

[9] Zhao QL, Gu ZH, Zeng HB. Resource synchronization and preemption thresholds within mixed-criticality scheduling. ACM Trans. on Embedded Computing Systems, 2015, 14(4): 81. [doi: 10.1145/2783440]

[10] Baruah S, Bertogna M, Buttazzo G. Multiprocessor Scheduling for Real-time Systems. Switzerland: Springer, 2015. 24–26. [doi: 10.1007/978-3-319-08696-5]

[11] Andersson B, Baruah S, Jonsson J. Static-priority scheduling on multiprocessors. In: Proc. of the 22nd IEEE Real-time Systems Symp. (RTSS 2001). London: IEEE, 2001. 193–202. [doi: 10.1109/REAL.2001.990610]

[12] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM, 1973, 20(1): 46–61. [doi: 10.1145/321738.321743]

[13] Joseph M, Pandya P. Finding response times in a real-time system. The Computer Journal, 1986, 29(5): 390–395. [doi: 10.1093/comjnl/29.5.390]

[14] Audsley N, Burns A, Richardson M, Tindell K, Wellings AJ. Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal, 1993, 8(5): 284–292. [doi: 10.1049/sej.1993.0034]

[15] Bini E, Buttazzo GC. Measuring the performance of schedulability tests. Real-time Systems, 2005, 30(1–2): 129–154. [doi: 10.1007/s11241-005-0507-9]

[16] Zhao QL. Resource synchronization protocols and design optimization techniques for mixed-criticality CPS [Ph.D. Thesis]. Hangzhou: Zhejiang University, 2015 (in Chinese with English abstract).

附中文参考文献:

[1] 乔磊, 龚健, 杨孟飞, 杨桦, 彭飞, 徐建, 谭彦亮, 吴一帆, 刘忠汉. 一种空间飞行器周期性混成随机任务调度方法: 中国, 201610814726.8. 2017-02-22.

[2] 杨茂林. 共享资源约束下的多核实时调度算法研究 [博士学位论文]. 成都: 电子科技大学, 2016.

[5] 马运南, 陈香兰. 支持SRP协议的实时调度技术. 计算机系统应用, 2016, 25(2): 1–8.

[16] 赵庆玲. 混合关键度CPS系统中的资源共享协议和设计优化 [博士学位论文]. 杭州: 浙江大学, 2015.

陈熙(1997－), 女, 硕士生, 主要研究领域为嵌入式操作系统, 实时调度, 形式化方法.

杨孟飞(1962－), 男, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究领域为空间飞行器嵌入式系统, 控制系统, 总体技术.

乔磊(1982－), 男, 博士, 研究员, CCF 杰出会员, 主要研究领域为操作系统模型设计, 存储管理, 文件系统.

刘洪标(1995－), 男, 博士生, 主要研究领域为嵌入式操作系统, 实时调度.