基于MPCP协议的任务最坏阻塞时间分析

曹永立，杨茂林，廖 勇

(电子科技大学信息与软件工程学院 成都 610054)

【摘要】多处理器天花板协议(MPCP)是经典的基于挂起机制的实时锁协议，被广泛应用于分组固定优先级(P-FP)调度下的多核/多处理器实时系统中。然而针对P-FP+MPCP调度的任务最坏阻塞时间分析往往过于保守，影响系统的可调度性。因此，该文提出一种计算实时任务最坏阻塞时间的新方法。其中实时任务模拟为非临界区与临界区的交替序列。该方法通过分析任务多次请求某一共享资源所需的最短执行时间，以及任务在任意时间内累计执行临界区时间的上限，提高了已有分析方法的计算准确性。可调度性实验表明，该方法优于已有方法，提高了系统可调度性。

关 键 词 阻塞；多核；分组调度；实时锁协议；实时系统

中图分类号 TP301.6 **文献标志码** A doi:10.3969/j.issn.1001-0548.2016.06.022

Worst-Case Blocking Time Analysis for MPCP

CAO Yong-li, YANG Mao-lin, and LIAO Yong

(School of Information and Software Engineering, University of Electronic Science and Technology of China Chengdu 610054)

Abstract Multiprocessor priority ceiling protocol (MPCP) is a classical suspension-based real-time locking protocol, wildly used in partitioned fixed-priority (P-FP) scheduled multiprocessor/multicore real-time systems. However, prior worst-case task blocking time (WCTBT) analysis is pessimistic, which negatively impacts the system schedulability. Therefore, a novel WCTBT analysis is proposed. In this analysis, a task is modeled to be an alternative sequence of normal and critical section segments. By analyzing the minimum execution time required for a task to request several shared resources, this method improves the accuracy of prior work and provides an upper bound on the cumulative execution time for a task to execute critical sections in any time interval. Schedulability experiments indicate that the proposed method outperforms the existing methods and improves the system schedulability significantly.

Key words blocking; multicores; partitioned scheduling; real-time locking protocols; real-time systems

在实时系统中，任务互斥访问关键数据结构、I/O设备等共享资源时，需要采用实时锁协议以避免死锁、阻塞链，同时减小优先级反转所造成的调度损失。在基于共享内存的多处理器实时系统中，文献[1]提出了一种单处理器优先级天花板协议(priority ceiling protocol, PCP)[2]的多处理器天花板协议(multiprocessor priority ceiling protocol, MPCP)。该协议适用于分组固定优先级(partitioned fixed-priority, P-FP)[3-4]调度下的多核/多处理器实时系统。

为了确保实时任务在其截止时限内完成，需要对任务的可调度性进行定量分析。在基于P-FP+MPCP的调度策略下，可调度性分析包含任务最坏阻塞时间分析。文献[1]将任务阻塞分为本地局部资源阻塞、本地全局资源阻塞、远程低优先级任务阻塞、远程高优先级任务阻塞，以及间接阻塞，并通过计算各部分阻塞时间的累加和得到任务最坏阻塞时间。由于前两个阻塞部分并非相互独立，该方法存在重复计算问题。文献[5]将任务描述为临界区与非临界区相间的模型，采用响应时间分析法(response time analysis, RTA)[6-8]计算任务远程阻塞时间。该方法减小了文献[1]中的重复计算问题，但没有考虑相邻临界区间的间隔，因此所得的最坏阻塞时间仍较保守。文献[3]利用任务中相邻临界区间的最短时间间隔，提出了新的任务最坏时间分析方法。该方法在相邻临界区最短时间间隔较大时可排除文献[1]中的部分保守计算，相反则可能得到更坏的结果。文献[9]改进了文献[5]中的计算方法，提高了任务最坏阻塞响应时间计算的准确性。为了进一步提高任务

收稿日期：2015－03－14；修回日期：2015－12－26
基金项目：国家自然科学基金(61103041)；国家863重大项目(2012AA010904)；中央高校基本科研业务费(ZYGX2012J070)
作者简介：曹永立(1992－)，男，主要从事信息工程方面的研究.

阻塞时间分析的准确性，本文结合文献[5]的任务模型以及文献[9]的RTA分析框架，提出了分析任务多次请求同一共享资源所需的最短执行时间，以及任务在任意时间内累计执行临界区上限的方法，并基于该分析方法提出一种新的任务最坏阻塞时间分析方法。该方法进一步减小了分析所得的任务最坏阻塞时间，实验表明，该分析方法可提高实时任务集合的可调度性。

1　任务模型

一组任务 $\Gamma=\{\tau_1,\tau_2,\cdots,\tau_n\}$ 在一个m核处理器上执行，并共享q个共享资源 $\Phi=\{\rho_1,\rho_2,\cdots,\rho_q\}$。 τ_i 的第l个作业表示为 $J_{i,l}$(一般情况下，用 J_i 表示 τ_i 的任意作业)。任务互斥访问共享资源，且一次最多允许获得一个共享资源。任务按优先级倒序编号，即若$i<j$则 τ_i 的优先级高于 τ_j。同一任务的所有作业的优先级相同(即 $J_{i,x}$ 和 $J_{i,y}$ 具有相同的优先级)。令 $\tau_{i,j}$ 表示 τ_i 的第j个非临界区段， $\tau_{i,j*}$ 表示 τ_i 的第j个临界区段。假设任务各段的执行时间确定，且各任务段按序依次执行，则 τ_i 可表示为(($NC_{i,1}$, $C_{i,1}$, $NC_{i,2}$, $C_{i,2}$,⋯), T_i)。其中， S_i 为 τ_i 的非临界区段数， $NC_{i,j}$ 和 $C_{i,j}$ 分别是 $\tau_{i,j}$ 和 $\tau_{i,j*}$ 的执行时间， T_i 为 τ_i 的最短到达时间间隔(或周期)。 τ_i 的执行时间可表示为 $C_i=\sum_{j=[1,Si-1]}(C_{i,j}+NC_{i,j})+NC_{i,Si}$。任务CPU利用率为 $u_i=C_i/T_i$，系统利用率为 $U=\sum_{\forall i}u_i$。本文假设任务相对截止时限等于任务周期。令 $J_{i,l}$ 的到达与结束时刻分别为 $r_{i,l}$ 和 $f_{i,l}$，则 τ_i 的最坏响应时间为 $R_i=\max_{\forall l}(f_{i,l}-r_{i,l})$。本文采用P-FP两级调度：首先将任务静态地划分到不同的处理器核上；然后在各核上根据任务的固定优先级进行抢占式调度。P-FP下，共享资源可分为本地资源和全局资源，前者为仅被分配到相同处理器核上的任务(简称同核任务)访问的资源，后者为可以被分配到不同处理器核上的任务(简称异核任务)访问的资源。MPCP采用PCP协议机制仲裁本地资源的共享。当任务访问某全局资源 ρ_k 而被阻塞时，该任务挂起并插入到 ρ_k 的优先级队列上等待。当该任务获得 ρ_k 时，其优先级将提升为 $\Omega_k=\pi_s+\pi_k$，直至释放 ρ_k。其中 π_s 为系统最高优先级(大于 τ_1 的优先级)， π_k 为需要访问 ρ_k 的最高优先级任务的优先级。根据MPCP，任务执行临界区时仍可能被抢占，使得任务的阻塞与抢占关系变得十分复杂[5, 9]。

2　任务请求共享资源时间分析

令 $N_{i,k}$ 表示 J_i 请求资源 ρ_k 的次数， $\tau_{i,k,x}$ 表示 τ_i 第x次访问 ρ_k 的临界区， $S_{i,k,x}$ 表示与 $\tau_{i,k,x}$ 对应的临界区编号。如图1所示， $N_{i,k}=2$， $N_{i,h}=1$， τ_i 在其第3个临界区中第2个请求 ρ_k，因此 $S_{i,k,2}=3$。从 $r_{i,l}$ 到 τ_i 第x次($1\leqslant x\leqslant N_{i,k}$)请求 ρ_k 的最短时间间隔可由式(1)计算如下：

$$d_{i,k,x}=\sum_{j=1}^{S_{i,k,x}}NC_{i,j}+\sum_{j=1}^{S_{i,k,x}-1}C_{i,j} \tag{1}$$

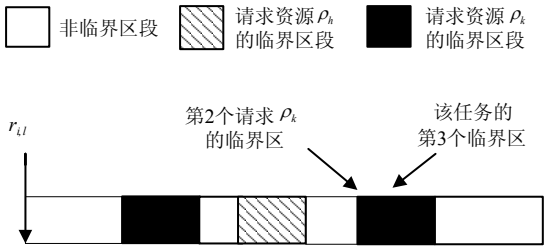

图1　任务模型示例，非临界区段与临界区段相互交错

表1　本文变量列表

变量	含义
$N_{i,k}$	J_i 请求资源 ρ_k 的次数
$\tau_{i,k,x}$	τ_i 第x次访问 ρ_k 的临界区
$S_{i,k,x}$	与 $\tau_{i,k,x}$ 对应的临界区编号
$d_{i,k,x}$	从 $r_{i,l}$ 到 τ_i 第x次($1\leqslant x\leqslant N_{i,k}$)请求 ρ_k 的最短时间间隔
$\delta_{i,k,x}(n)$	从 J_i 开始执行 $\tau_{i,k,x}$ 起，到其随后第n次请求 ρ_k 的最短时间间隔
$\eta_{i,k,x}(\Delta t)$	从 $J_{i,l}$ 执行 $\tau_{i,k,x}$ 开始， τ_i 在 Δt 时间内请求 ρ_k 的最大次数
$\psi_{i,k,x}(n)$	从 $J_{i,l}$ 执行 $\tau_{i,k,x}$ 开始， τ_i 在接下来的n次对 ρ_k 的请求中其相应的临界区执行时间
$\xi_{i,k}$	τ_i 的一个作业在与 ρ_k 相应的临界区中的累计执行时间
$\varphi_{i,k}$	τ_i 在 ρ_k 的临界区内被其他任务抢占的最大时间
$\varepsilon_{i,k}$	τ_i 占用资源 ρ_k 的最大时间
$b_{i,k,L}$	τ_i 每次被异核低优先级任务阻塞的时间上限
$b_{i,k,H}(\Delta t)$	τ_i 每次请求全局资源 ρ_k 时，在 Δt 内被异核高优先级任务阻塞的最大时间
$Y_{i,k}$	τ_i 每次请求全局资源时，被远程阻塞的最大时间
Y_i	τ_i 最大远程阻塞时间
$\delta_{i,*,j}(n)$	从 J_i 开始执行 $\tau_{i,k,x}$ 起，执行n个临界区的最短时间间隔
$\psi_{i,*,j}(n)$	从 $J_{i,l}$ 执行 $\tau_{i,k,x}$ 开始，执行n个临界区的临界区执行时间之和
$B_{i,L}$	在 τ_i 的一次执行中，其被本地低优先级任务阻塞的时间上限
B_i	τ_i 的最大本地阻塞时间

定理 1　令 t_x 为 $J_{i,l}$ 开始执行 $\tau_{i,k,x}$($1\leqslant x\leqslant N_{i,k}$)

的时刻，t_n 为该任务从 t_x 开始到第n次($n\geqslant 1$，且将 $J_{i,l}$ 执行 $\tau_{i,k,x}$ 设为第一次)请求资源 ρ_k 的时刻，则 $\min(t_n-t_x)=\delta_{i,k,x}(n)$。

其中，当 $x\leqslant N_{i,k}-n+1$ 且 $1\leqslant n\leqslant N_{i,k}$ 时：

$$\delta_{i,k,x}(n)=d_{i,k,(x+n-1)}-d_{i,k,x} \tag{2a}$$

当 $x>N_{i,k}-n+1$ 且 $1\leqslant n\leqslant N_{i,k}$ 时：

$$\delta_{i,k,x}(n)=C_i-d_{i,k,x}+T_i-R_i+d_{i,k,(x+n-1-N_{i,k})} \tag{2b}$$

当 $x+n-1>2N_{i,k}$ 时：

$$\delta_{i,k,x}(n)=(\lceil (n+x-1)/N_{i,k}\rceil-1)T_i-d_{i,k,x}+d_{i,k,f(x)}+C_i-R_i \tag{2c}$$

其中，

$$f(x)=n+x-1-(\lceil (n+x-1)/N_{i,k}\rceil-1)N_{i,k}$$

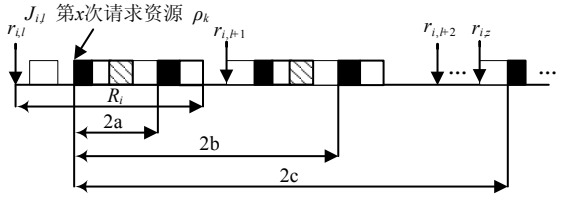

图2 从 $J_{i,l}$ 第x次请求资源 ρ_k 开始，请求n次 ρ_k 所需时间示例

证明：当 $x\leqslant N_{i,k}-n+1$ 且 $1\leqslant n\leqslant N_{i,k}$ 时，$x+n-1\in[x,N_{i,k}]$，即n次请求在 τ_i 的一个作业内完成，如图2所示。根据式(1)可知，$\delta_{i,k,x}(n)=d_{i,k,(x+n-1)}-d_{i,k,x}$。

当 $x>N_{i,k}-n+1$ 且 $1\leqslant n\leqslant N_{i,k}$ 时，$x+n-1\in[N_{i,k},2N_{i,k}]$，即$n$次请求由两个连续作业 $J_{i,l}$ 和 $J_{i,l+1}$ 完成。其中，$[t_x,t_n]$ 可分为三个部分：$[t_x,f_{i,l}]$，$[f_{i,l},r_{i,l+1}]$，$[r_{i,l+1},t_n]$。若 $J_{i,l}$ 从 t_x 起连续执行(不被中断)直至任务完成，则 $[t_x,f_{i,l}]$ 取得最小值 $C_i-d_{i,k,x}$。由于 $R_i=\max_{\forall f}(f_{i,l}-r_{i,l})$，所以当 $f_{i,l}-r_{i,l}=R_i$ 时，$r_{i,l+1}-f_{i,l}=r_{i,l+1}-r_{i,l}-R_i=T_i-R_i$ 取得最小值，即 $[f_{i,l},r_{i,l+1}]$ 的最小值为 T_i-R_i。由于 $J_{i,l}$ 已完成 $N_{i,k}-x+1$ 个请求，$J_{i,l+1}$ 需完成剩余的 $n-N_{i,k}+x-1$ 个请求。根据式(1)，$[r_{i,l+1},t_n]$ 的最小值为 $d_{i,k,(x+n-1-N_{i,k})}$。综合以上分析，式(2b)得证。

当 $x+n-1>2N_{i,k}$ 时，n次请求由至少三个连续作业完成。令 $J_{i,z}$ 表示完成n次请求的最后的一个作业，则 $[t_x,t_n]$ 可分为三个部分：$[t_x,r_{i,l+1}]$，$[r_{i,l+1},r_{i,z}]$，$[r_{i,z},t_n]$。根据前述分析可知，$[t_x,r_{i,l+1}]$ 的最小值为 $C_i-d_{i,k}^x+T_i-R_i$。$[r_{i,l+1},r_{i,z}]$ 可包含 $\lceil (n-N_{i,k}+x-1)/N_{i,k}\rceil-1$ 个完成周期，因此该段时间为 $(\lceil (n+x-1)/N_{i,k}\rceil-2)T_i$。$[r_{i,z},t_n]$ 为 $J_{i,z}$ 执行直至完成余下的 $n-N_{i,k}+x-1-(\lceil (n+x-1)/N_{i,k}\rceil-2)N_{i,k}$ 个请求所需的最短时间。根据$f(x)$定义及式(1)可知，$[r_{i,z},t_n]$ 的最小值为 $d_{i,k,f(x)}$。综合以上分析，式(2c)得证。

定理1给出了一个任务从某个时刻开始任意次请求某个共享资源所需的最短时间。通过求 $\delta_{i,k,x}(n)$ 的反函数，可计算出任意时间间隔内一个任务请求某个共享资源的最大次数。

推论 1 从 $J_{i,l}$ 执行 $\tau_{i,k,x}$ $(1\leqslant x\leqslant N_{i,k})$ 的时刻开始，τ_i 在 Δt 时间内请求 ρ_k 的最大次数为 $\eta_{i,k,x}(\Delta t)=\max\{n\mid\delta_{i,k,x}(n)\leqslant\Delta t\}$。

参照 $\delta_{i,k,x}(n)$ 的计算方法，还可推导出一个任务请求任意次某共享资源时在相应的临界区内执行的时间。进而可计算出任意时间间隔内，一个任务在与某共享资源相应的临界区内的执行时间上限。

推论 2 从 $J_{i,l}$ 执行 $\tau_{i,k,x}$ $(1\leqslant x\leqslant N_{i,k})$ 的时刻开始，τ_i 在接下来的n次对 ρ_k 的请求中其相应的临界区执行时间为 $\psi_{i,k,x}(n)$。

其中，当 $x\leqslant N_{i,k}-n+1$ 且 $1\leqslant n\leqslant N_{i,k}$ 时：

$$\psi_{i,k,x}(n)=\sum_{j=x}^{x+n-1}C_{i,j} \tag{3a}$$

当 $x>N_{i,k}-n+1$ 且 $1\leqslant n\leqslant N_{i,k}$ 时：

$$\psi_{i,k,x}(n)=\xi_{i,k}-\sum_{j=x+n-N_{i,k}}^{x-1}C_{i,j} \tag{3b}$$

当 $x+n-1>2N_{i,k}$ 时：

$$\psi_{i,k,x}(n)=\xi_{i,k}\lfloor n/N_{i,k}\rfloor+\psi_{i,k,x}(n-N_{i,k}\lfloor n/N_{i,k}\rfloor) \tag{3c}$$

式中，$\xi_{i,k}=\sum_{\Phi(\tau_{i,j^*})=\rho_k}C_{i,j}$ 是 τ_i 的一个作业在与 ρ_k 相应的临界区中的累计执行时间(令 $\Phi(\tau_{i,j^*})$ 为临界区 τ_{i,j^*} 对应的共享资源)。

推论 3 从 $J_{i,l}$ 执行 $\tau_{i,k,x}$ $(1\leqslant x\leqslant N_{i,k})$ 的时刻开始，τ_i 在 Δt 时间内在与 ρ_k 相应的临界区上的执行时间上限为 $\psi_{i,k,x}(\eta_{i,k,x}(\Delta t))$。

3 任务最坏阻塞时间分析

P-FP调度下，任务 τ_i 请求共享资源时可能发生本地阻塞(被本核任务阻塞)或远程阻塞(被异核任务阻塞)。当 τ_i 被远程阻塞时，锁住相应资源的异核任务可能在临界区内被抢占，从而可能对 τ_i 造成传递阻塞。本节采用文献[9]的任务阻塞时间分析框架，结合以上时间分析结果，提出新的任务最坏阻塞时

间分析方法。

3.1 远程阻塞时间

由于各全局资源的优先级天花板可能不同，任务在临界区内仍可能被抢占。如图3所示，由于 $\Omega_h=\pi_s+\pi_1>\Omega_k=\pi_s+\pi_2$，$J_2$ 在 ρ_k 的临界区内被 J_1 在 t_2 时刻抢占，被 J_3 在 t_3 时刻抢占。根据文献[9]，任务 τ_i 在 ρ_k 的临界区内被其他任务抢占的最大时间为：

$$\varphi_{i,k}=\sum_{\tau_a\in\Gamma_{i,\text{local}}}\max_{j\in[1,S_{a-1}]\wedge\Phi(\tau_{a,j^*})=\rho_k\wedge\Omega_h>\Omega_k}C_{a,j} \tag{4}$$

式中，$\Gamma_{i,\text{local}}$ 为 τ_i 的本地任务。进而可知，τ_i 占用资源 ρ_k 的最大时间为 $\varepsilon_{i,k}=\varphi_{i,k}+\max_{j\in[1,S_{i-1}]\wedge\Phi(\tau_{i,j^*})=\rho_k}C_{i,j}$。如图3所示，$\varepsilon_{2,k}=t_4-t_1$。

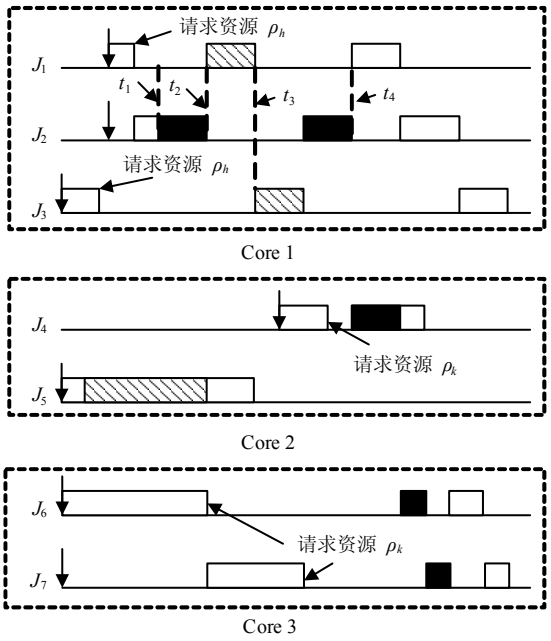

图3　远程阻塞示例

当任务 τ_i 请求全局资源 ρ_k 时，可能被异核低优先级任务阻塞。由于MPCP采用优先级排队机制，因此 τ_i 每次只能被一个低优先级任务阻塞。其中，τ_i 每次被异核低优先级任务阻塞的时间上限为：

$$b_{i,k,L}=\max_{\tau_l\in\Gamma-\Gamma_{i,\text{local}}\wedge l>i}\varepsilon_{l,k} \tag{5}$$

当 τ_i 在某全局资源的优先级队列中等待时，若有高优先级异核任务请求相同资源，则这些异核任务会插入到 τ_i 前面，先于 τ_i 获得相应全局资源。如图3所示，J_6 在 J_4 之前请求 ρ_k，而由于 J_4 优先级较高，J_4 先于 J_6 获得 ρ_k。值得注意，在 τ_i 挂起等待 ρ_k 期间，异核高优先级任务可能多次请求 ρ_k，从而可能多次阻塞 τ_i。当某异核高优先级任务 τ_h 开始执行 $\tau_{h,k,x}$ $(1\leqslant x\leqslant N_{i,k})$ 时，在 Δt 时间内，τ_h 在 ρ_k 相应临界区内的最大执行时间为 $\psi_{h,k,x}(\eta_{h,k,x}(\Delta t))$，在临界区内被抢占的最大时间为 $\eta_{h,k,x}(\Delta t)\varphi_{i,k}$。因此 τ_i 每次请求全局资源 ρ_k 时，在 Δt 内被异核高优先级任务阻塞的最大时间为：

$$b_{i,k,H}(\Delta t)=\sum_{\tau_h\in\Gamma-\Gamma_{i,\text{local}}\wedge h<i}\max_{x\in[1,N_{i,k}]}(\psi_{h,k,x}(\eta_{h,k,x}(\Delta t)))+\eta_{h,k,x}(\Delta t)\varphi_{i,k}) \tag{6}$$

由于式(5)等号两边关于 Δt 单调递增，且 τ_i 的远程阻塞时间为 τ_i 被异核低优先级任务和异核高优先级任务阻塞的时间之和，因此 τ_i 每次请求全局资源时，被远程阻塞的时间可通过以下迭代过程决定：

$$Y_{i,k}^{n+1}=b_{i,k,L}+b_{i,k,H}(Y_{i,k}^{n}) \tag{7}$$

式中，$Y_{i,k}^{0}=0$，当 $Y_{i,k}^{n+1}=Y_{i,k}^{n}$，$Y_{i,k}=Y_{i,k}^{n+1}$；若 $Y_{i,k}^{n+1}>T_i$，则迭代可提前结束，此时 τ_i 不可调度。令 $\Phi_{i,G}$ 表示 τ_i 请求的全局资源集合，根据式(7)可得 τ_i 的最大远程阻塞时间为：

$$Y_i=N_{i,k}\sum_{\rho_k\in\Phi_{i,k}}Y_{i,k} \tag{8}$$

3.2 本地阻塞时间

在任务 τ_i 就绪前或被远程阻塞而挂起时，其本地低优先级任务可能被调度执行并请求共享资源。当低优先级本地任务获得并继承相应资源的优先级天花板时，其有效优先级将高于 τ_i，从而发生优先级翻转，延迟 τ_i 的执行。在实时系统中，这种由优先级翻转引起的任务延迟被视作任务阻塞[10]。如图4所示，在 J_1 挂起等待全局资源 ρ_k 期间，J_2 和 J_3 相继请求全局资源 ρ_k 并挂起等待。此后，当 J_1 执行非临界区时，J_2 和 J_3 的共享资源请求相继得到满足，从而阻塞 J_1。

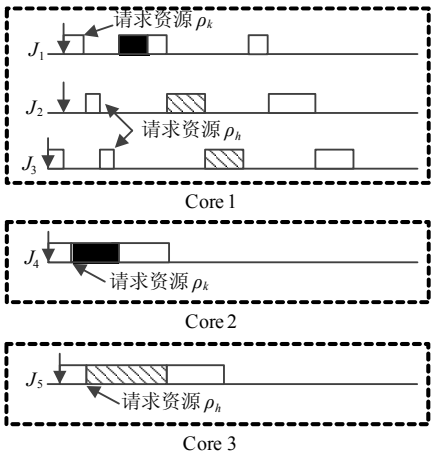

图4　本地阻塞示例

令 $N_{i,G}$ 为 τ_i 的全局临界区数，则 τ_i 最多因远程阻塞而挂起 $N_{i,G}$ 次。此外，由于在 τ_i 到达时低优先

级任务可能已经获得共享资源，因此 τ_i 最多可被每个低优先级本地任务阻塞 $N_{i,G}+1$ 次。设 τ_l 为 τ_i 的本地低优先级任务，τ_l 只有执行临界区时才可能阻塞 τ_i，而在 τ_i 就绪后，τ_l 只有在 τ_i 挂起期间才有机会执行其非临界区，因此 τ_l 在 τ_i 就绪期间的执行时间减去 τ_i 的远程阻塞时间即为 τ_l 对 τ_i 的本地阻塞时间。

设 t_{x^*} 为 $J_{i,l}$ 开始执行第 $j\,(1\leqslant j\leqslant S_{i-1})$ 个临界区的时刻，t_{n^*} 为该任务从 t_{x^*} 开始到 τ_i 第 n 次进入临界区的时刻。令 $\delta_{i,*,j}(n)=\min(t_{n^*}-t_{x^*})$，令 $\psi_{i,*,j}(n)$ 表示 τ_i 从 t_{x^*} 起执行的 n 个临界区时间之和。其中，$\delta_{i,*,j}(n)$ 和 $\psi_{i,*,j}(n)$ 的计算方法与定理1和推论2类似(推论1、2针对特定共享资源 ρ_k 的临界区，而 $\delta_{i,*,j}(n)$ 和 $\psi_{i,*,j}(n)$ 针对连续临界区)。根据以上分析，$\delta_{i,*,j}(n)=\psi_{i,*,j}(n)+Y_i$。由于 $\delta_{i,*,j}(n)\leqslant\psi_{i,*,j}(n)+Y_i$，因此 $\psi_{i,*,j}(n)\geqslant\delta_{i,*,j}(n)-Y_i$。从而，在 τ_i 的一次执行中，其被本地任务阻塞的时间上限为：

$$B_{i,L}=\max_{x\in[1,S_{i-1}]\wedge n>0}\{\psi_{i,*,j}(n)\,|\,\psi_{i,*,j}(n)\geqslant\delta_{i,*,j}(n)-Y_i\} \quad (9)$$

由于在任务 τ_i 就绪前或被远程阻塞而挂起时，其所有本地低优先级任务均可能先后被调度执行并请求共享资源，因此 τ_i 的本地阻塞时间不大于所有本地低优先级任务对请求造成的最大阻塞之和，即：

$$B_i=\sum_{l>i\wedge\tau_l\in\Gamma_{i,\mathrm{local}}}B_{i,L} \quad (10)$$

4 可调度性实验

本节实验在Microsoft VS平台上进行，根据UUnifast-Discard算法[11]随机产生任务集合。其中，任务周期在[200,1 000]内服从均匀分布，u_i 在[0.05,0.2]内随机产生，C_i根据任务CPU利用率定义由 u_i 与 T_i 决定。每个任务有$x\in$[1,10]个临界区，各临界区长度$y\in$[0.5,8]。系统中有q=20个共享资源，任务各临界区访问各共享资源的概率均等，且每个任务集合中最多有20个任务访问同一个共享资源。

将以上所得任务集合按照Worst-Fit算法(以避免任务分配故障问题[12])进行任务分配(本文模拟m=8核处理器)，并采用Raj[1]、KDR[5]、YLLH[9]，以及本文提出的方法(Proposed)计算任务最坏阻塞时间。根据各方法所得最坏阻塞时间，采用文献[5]的RTA方法分析任务集合的可调度性。每次实验随机产生h=5 000个任务集合，分别计算以上4种分析方法的可调度率α。其中，可调度率计算方法如下：设一次实验随机产生c个任务集合，其中由分析方法A得出有g个任务集合可调度，则方法A的可调度率为 $\alpha=g/c$。由于以上各方法的所得的任务最坏阻塞时间不相同，而实验中采用了相同的可调度分析算法，因此，实验结果中各方法的可调度率反映了最坏阻塞时间的差异对的系统可调度性的影响。可调度率越高的方法，其分析所得的最坏阻塞时间越小。

图5中，每个任务有4个临界区，临界区长度为1.5。该实验显示，各方法的可调度率均随系统利用率增加而下降，其中新方法的可调度率高于其余方法。新方法通过计算任务执行多个临界区所需的执行时间下限以及在任意时间内执行临界区的时间上限，将给定时间内不可能发生的阻塞时间排除在分析所得的最坏阻塞时间以外。而其余方法对任务在某段具体时间内是否会发生任务阻塞不敏感，使得分析结果较保守。

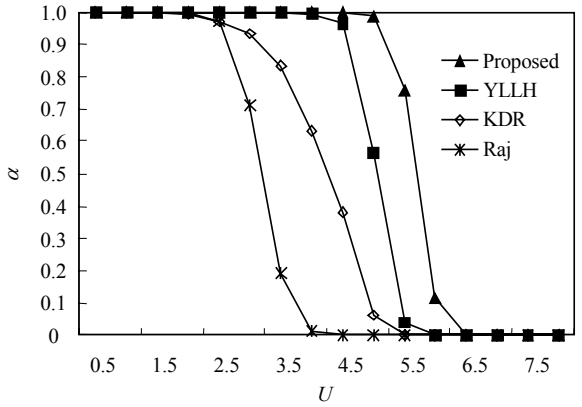

图5 $x=4$，$y=1.5$时α与U的关系

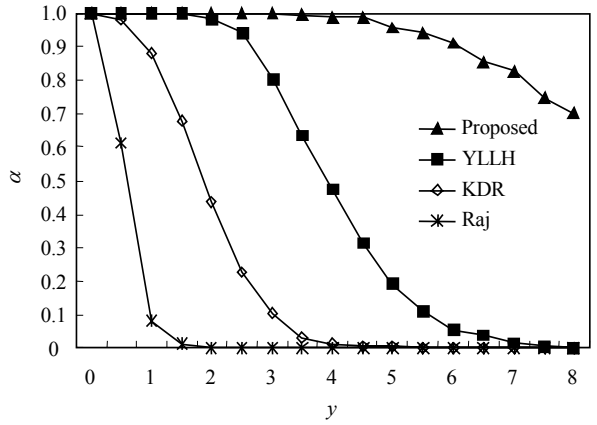

图6 $x=4$，$U=4$时α与y的关系

图6中，系统利用率为4，每个任务有4个临界区。由于任务阻塞时间随临界区长度增加而增加，各分析方法的可调度率均随临界区长度增加而下降。当 $y\leqslant3.5$ 时Proposed的可调度率为1；而在y=3.5时，YLLH与KDR的可调度率分别约为0.62和0.03，Raj为0。当y>3.5时，Proposed的可调度率下降速度低于

其余方法。图7中，系统利用率为4.5，任务临界区长度为1.5。随任务的临界区数增加，任务在最坏情况下被阻塞的次数随之增加，从而降低了可调度率。其中，各方法的α值随x的变化趋势与图6相似，Proposed的可调度率明显高于其余方法。

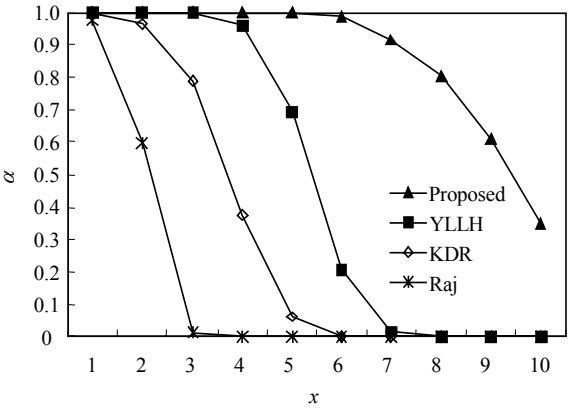

图7　$y=1.5$，$U=4.5$时α与x的关系

5 结 束 语

本文针对P-FP+MPCP调度，采用文献[5]的任务模型，将任务模拟为非临界区与临界区的交替序列，并对任务访问共享资源的时间关系进行定量分析。通过分析得出任务在任意时间内执行临界区段的时间之和，从而得出任意任务在给定时间内对其他任务造成的最大阻塞。在此基础上，结合文献[9]的阻塞时间分析框架，提出一种新的任务最坏阻塞时间分析方法。可调度性实验结果显示，新的分析方法明显优于已有方法，提高了系统的可调度性。

本文的研究工作得到华为基金(IRP- 2012-02-07)及开源项目www.aCoral.org的资助，在此表示感谢。

参 考 文 献

[1] RAJKUMAR R. Real-time synchronization protocols for shared memory multiprocessors[C]//IEEE International Conference on Distributed Computing Systems. Paris: IEEE, 1990: 116-123.

[2] SHA L, RAJKUMAR R, LEHOCZKY J. Priority inheritance protocols: an approach to real-time synchronization[J]. IEEE Transactions on Computers, 1990, 39(9): 1175-1185.

[3] SCHLIECKER S, NEGREAN M, ERNST R. Response time analysis on multicore ECUs with shared resources[J]. IEEE Transactions on Industrial Informatics, 2009, 5(4): 402-413.

[4] BRANDENBURG B. Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling[C]// IEEE International Conference on Real-Time Embedded Technology and Application Symposium. Philadelphia: IEEE, 2013: 141-152.

[5] LAKSHMANAN K, NIZ D, RAJKUMAR R. Coordinated task scheduling, allocation and synchronization on multiprocessors[C]//IEEE International Conference on Real-Time System Symposium. Washington D. C: IEEE, 2009: 469-478.

[6] AUDSLEY N, BURNS A, RICHARDSON M, et al. Applying new scheduling theory to static priority preemptive scheduling[J]. Journal of Software Engineering, 1993, 8(5): 284-296.

[7] BINI E, BUTTAZZO G. Schedulability analysis of periodic fixed priority systems[J]. IEEE Transactions on Computers, 2004, 53(11): 1462-1473.

[8] BRIL R, LUKKIEN J, VERHAEGH W. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption[J]. Real-Time Systems, 2009, 42(1-3): 63-119.

[9] YANG Mao-lin, LEI Hang, LIAO Yong. Synchronization analysis for hard real-time multicore systems[J]. Applied Mechanics and Materials, 2012, 241-244: 2246-2253.

[10] BRANDENBURG B, ANDERSON J. Optimality results for multi-processor real-time locking[C]//IEEE International Conference on Real-Time Systems Symposium. San Diego: IEEE, 2010: 49-60.

[11] DAVIS R, BURNS A. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems[J]. Real-Time Systems. 2011, 47(1): 1-40.

[12] 刘加海, 杨茂林, 雷航, 等. 共享资源约束下多核实时任务分配算法研究[J]. 浙江大学学报(工学版), 2014, 48(1): 113-117.
LIU Jia-hai, YANG Mao-lin, LEI Hang, et al. A research for multicore real-time task allocation algorithms[J]. Journal of Zhejiang University(Engineering Science), 2014, 48(1): 113-117.

编　辑　蒋　晓