# 新的混合关键任务调度算法的研究

朱怡安，黄姝娟，段俊花，陆 伟

(西北工业大学计算机学院 西安 710072))

【摘要】分析了混合关键系统中当前任务调度方法存在的问题，提出一种正反向时间分割和关键因子优先的调度算法，该方法将所有混合关键任务按照不同级别进行时间正反向分割得出空闲时间窗口，并根据关键因子的大小决定优先级别。为了减少在关键级别转化过程中优先级低的任务丢失死限的工作数量，将空闲窗口分配给由于关键级别转化而使得优先级落后的任务。仿真实验表明，该方法在降低任务丢失死限率和完成任务的数量方面比按照关键级别进行优先级指派算法(CAPA)和OCBP方法较优。

关 键 词 算法; 关键因子; 混合关键任务; 实时系统; 调度
中图分类号 TP319 文献标志码 A doi:10.3969/j.issn.1001-0548.2014.02.021

# New Scheduling Algorithm for Mixed-Criticality Real-Time Task Sets

ZHU Yi-an, HUANG Shu-juan, DUAN Jun-hua, and LU Wei

(School of Computer Science, Northwestern Polytechnical University Xi'an 710072)

**Abstract** By analyzing the problems of real-time scheduling in mix-criticality systems, a new scheduling algorithm, forward and backward time window partition-criticality factor prior (FBTWP-CFP), is proposed. It can separate the running time windows for the tasks of all criticality levels offline from the forward direction and backward direction and generate the idle windows for the tasks which get the lower priority by the criticality factor for the criticality level changed. The simulation results show that FBTWP-CFP is better than criticality assigned priority algorithm (CAPA) and own criticality based priority (OCBP) in completed tasks number and reduced miss deadline ratio.

**Key words** algorithm; criticality factor; mixed-criticality tasks; real-time system; scheduling

随着越来越多的嵌入式系统采用片上多处理器技术，为了节省能源，设计者将更多的安全关键任务集成在同一个平台下，称该系统为混合关键系统(mixed-criticality system)。这些安全关键任务称为混合关键任务(mixed-criticality tasks)[1]。这种系统在不同时刻，任务将处于不同的安全关键级别。要想让各项任务都能够根据自己的安全关键级别合理、顺利地得到执行，就必须对各安全关键级别下的所有任务统一进行调度。这种多核平台下对混合关键级别任务的调度(mixed criticality schedulable，MCS)就成为焦点问题[2-3]。

## 1 相关工作

目前关于混合关键任务的调度算法大致分为3类[4]：一类是基于资源保留(reservations-based scheduling)的调度方法，该方法是直接将MC的任务转化为带有相同释放时间、死限时间以及处理器执行时间的非MC任务，再用传统的调度方法来调度。如WCRS(worst-case reservations schedulable)和最早虚拟死限优先算法(earliest deadline first-virtual deadline，EDF-VD[5])。WCRS是保留最坏情况下的执行时间的调度方法，该方法使得系统利用率较低。而EDF-VD算法是为了求得最大利用率，根据系统本身所处的安全关键级别以及每个任务的安全关键级别，实时地为各任务设置相应的虚拟死限，从而根据EDF原则对各个任务进行实时调度。该方法的缺点是在满足任务死限不丢失的情况下对处理器处理速度要求很高。

第二类是基于优先级的调度方法(priority- based scheduling)，如CAPA和OCBP[6]调度。CAPA是将关键级别高和低的任务分别指定为优先级高和低的任务。该算法的缺点是当系统不处于高关键级别时，

收稿日期：2012－12－21； 修回日期：2013－01－20
基金项目：航空科学基金(20130753006)；航天科技创新基金(2011XR160001)
作者简介：朱怡安(1961－)，男，博士，教授，主要从事嵌入式与普适计算、并行计算等方面的研究.

低关键级别的任务没有办法被公平调度，有时会导致优先级翻转现象[7]。OCBP是根据每个任务本身的关键系数(包括每个级别下的最坏执行时间、任务的发布时间、运行周期等)为各个任务预先设置优先级，然后根据优先级对已发布的各个任务进行调度。其操作方法是在选择正确的优先级顺序时，先选择最低优先级任务，在每一步选择一个较低优先级任务时，如果调度可以顺利进行，就制定它为当前优先级然后转向下一个较高优先级；否则，另外一个任务被选择为当前的优先级。当所有任务都被指派上优先级或者当没有剩余的任务在当前的优先级可以被调度时，该算法终止。该算法的缺点是如果某次所有任务的最低优先级设定都失败了，该算法就无法进行下去，且根据关键级别来指派优先级的调度方法将会导致较低的系统利用率。

第三类是slack-aware方法。文献[8]指出当前硬实时调度和分析技术不能够有效利用多核平台提供的计算带宽，原因是在预计的最坏执行时间和实际执行时间之间有很大的差距。将该差距称为“Slack”。在调度分析时可以计算并重新分配给关键级别低的任务。该文献在多核平台下给出了一个调度体系结构来重新分配Slack却没有说明Slack的计算方式以及如何实现。文献[1]在CAPA方法的基础上计算了从关键模式到正常模式Slack发现算法，减少了关键级别翻转(criticality inversion)现象而且能保留合理的利用率，但该方法要求对每一个任务的Slack时间进行多次推算，计算过程很繁琐，时间复杂度高。文献[9]对Zero-Slack和优先级指派以及周期转换方法进行了评价，指出优先级指派和周期转换方法更容易能调度一个随机的任务集，而Zero-Slack方法只能调度一小部分的任务集。

本文结合资源保留方法和优先级方法的优点，提出了一种基于正反向时间分割和关键因子优先的调度方法。该方法是一种比较简单、有效的方法，不仅能有效地保证利用率而且还能减少关键级别翻转现象和任务死限丢失的现象。

## 2　混合安全关键任务模型

### 2.1　job模型

在一个共有$K$个级别的混合安全关键系统中，一个job可以用一个4维向量表示为$\boldsymbol{J}_j=(r_j,d_j,x_j,\boldsymbol{c}_j)$，其中，$r_j$表示该job的到达时间，也可以成为发布时间；$d_j$表示任务的死限时间，$d_j>r_j$；$x_j\in\{1,2,\cdots,K\}$，表示该job在某一时刻的关键级别，在某一确定的时刻，它的值也是确定的；$\boldsymbol{c}_j$代表一个$K$维向量，即$\boldsymbol{c}_j=\{c_j(1),c_j(2),\cdots,c_j(K)\}$，表示该job在每个级别对应的最坏执行时间，在这里假设$c_j(1)\leqslant c_j(2)\leqslant\cdots\leqslant c_j(K)$。对于一个任务集合中的每一个job，要想确保实时系统的特性，其执行时间必须在死限到来之前结束，也就是说，实际执行时间结束的时刻要小于死限时刻。

### 2.2　task模型

在混和安全关键任务模型中，每一个task可以用一个3维向量表示为$\boldsymbol{T}_j=(\boldsymbol{c}_j,p_j,x_j)$。其中，$\boldsymbol{c}_j$同job模型中的一样，是一个$K$维向量，表示该task处于各个级别下时对应的最坏执行时间，即$\boldsymbol{c}_j=\{c_j(1),c_j(2),\cdots,c_j(K)\}$；$p_j$表示该task的执行周期，即每一个task$\boldsymbol{T}_j$可以隐含无数个job，这些job能够每隔$p_j$的间隔发布并执行一次，其死限时间就规定为该job发布之后经过$p_j$之后的时刻，即$d_j=r_j+p_j$。$x_j$同job模型一样，表示该task所处的级别，$x_j\in\{1,2,\cdots,K\}$，在某一确定的时刻，它的值也是确定的。同样，在task模型中，针对$\boldsymbol{c}_j$，假设$c_j(1)\leqslant c_j(2)\leqslant\cdots\leqslant c_j(K)$，job的$\boldsymbol{c}_j$与该task的$\boldsymbol{c}_j$同样保持一致。

### 2.3　系统模型

在一个混合安全关键任务系统中，假设共有$K$个安全关键级别，设$T=\{T_1,T_2,\cdots,T_m\}$表示该系统中的任务集合，其中，$T_i=(c_i,p_i,x_i)$，$i\in(1,m)$，其中$c_i$表示该任务的最坏执行时间；$p_i$表示该任务的执行周期；$x_i$表示该任务在此混合安全关键任务系统中的关键级别。

## 3　FBTWP-CFP算法设计与实现

### 3.1　相关定义

**定义 1**　最早可以开始执行时刻，指$J_i$可以最早获得空闲处理器的时刻，用$S_i^e$表示。

**定义 2**　最晚开始执行时刻，指$J_i$的绝对死限减去其所需要执行的最坏执行时间所得到的时刻值。用$S_i^l$表示，即$S_i^l=d_i-c_i$。

**定义 3**　最晚执行窗口，指计算$J_i$最晚开始执行时刻，从最晚开始执行的时刻到绝对死限这一段时间为其最迟执行窗口。用$W_i^l$表示，即$W_i^l=[d_i-c_i,d_i]$。

**定义 4**　最早执行完毕时刻，指计算$J_i$最早可以执行时刻加上其所需要的最大执行时间所得到的值。用$O_i$表示，即$O_i=S_i^e+c_i$。

**定义 5** 最早执行窗口，指 $J_i$ 从最早可以执行时刻到最早执行完毕时刻这一段时间。用 $W_i^e$ 表示即 $W_i^e=[S_i^e,O_i]$。

**定义 6** 空闲可执行窗口，指如果 $J_i$ 的最早可以执行完毕时刻 $O_i$ 大于最晚开始执行时刻 $S_i^l$ 的这段时间就称为空闲可执行窗口。用 $W_i^I$ 表示，即 $W_i^I=[O_i,S_i^l]$。

**定义 7** 相对关键度，指一个job $J_i$ 相对于其他job的关键级别重要程度，用 $\rho_i=x_i\Big/\sum_{j=1}^{n}x_j$ 表示。

**定义 8** 关键额度，指一个job $J_i$ 在$K$个关键级别下所能具有的关键份数，用 $\delta_i=x_i/K$ 表示。

**定义 9** 关键因子，指一个job $J_i$ 在某个级别下的一次执行相对于其他任务的一次执行所处关键级别重要程度以及利用率的一种度量。用 $\theta_i^k=\rho_i\delta_i u_i^k$ 表示，且 $0\leqslant\theta_i^k\leqslant 1$，其中，$u_i^k$ 为 $J_i$ 在 $k$ 级别下的利用率。

**定理** 假设在同一个核上进行调度且不考虑上下文切换和中断等额外开销的情况下，某种调度方案对于同时刻到达的任务集，工作 $J_i$ 是否可以被调度成功当且仅当该工作的周期要大于或等于其前面已经执行过的任务的执行时间与该任务需要执行的时间值之和。

证明：1) 必要性证明。令某工作集 $J$ 执行顺序为 $J_1,J_2,\cdots,J_n$，设每个任务执行完毕的时刻为 $a_1,a_2,\cdots,a_n$，若所有任务都能满足死限，由于不计算额外开销，那么前面工作的释放时间就是后面任务的开始执行时间。则必须：

$$\begin{gathered}a_1=r_1+c_1\leqslant d_1\\ a_2=a_1+c_2=r_1+c_1+c_2\leqslant d_2\\ a_3=a_2+c_3=r_1+c_1+c_2+c_3\leqslant d_3\\ \vdots\\ a_n=a_{n-1}+c_n=r_1+c_1+c_2+\cdots+c_n\leqslant d_n\\ d_n=p_n+r_n\end{gathered}$$

因为是同时刻到达，因此 $r_n=r_1$，结合上式可以得到：

$$r_1+c_1+c_2+\cdots+c_n\leqslant r_n+p_n\Rightarrow\sum_{i=1}^{n}c_i\leqslant p_n$$

得证。

2) 充分性证明。如果每个工作的周期都大于其前面执行任务的执行时间之和，那么任何一个工作周期大于该工作的执行时间与其前面执行过的工作的执行时间之和，则有：

$$\begin{gathered}p_i\geqslant c_i+c_{i-1}+c_{i-2}+\cdots+c_1\\ \Rightarrow d_i=r_i+p_i\geqslant r_i+c_i+c_{i-1}+c_{i-2}+\cdots+c_1\end{gathered}$$

表明任何一个任务执行都没有超过死限，所以是可以被调度的。此定理得证。

那么对于非同时到达的工作则判定其是否可被调度的条件分为两种情况：

1) 当满足 $r_i\geqslant a_{i-1}$，$i=2,3,\cdots,n$ 时，有：

$$\begin{gathered}a_1=r_1+c_1\leqslant d_1=r_1+p_1\\ a_2=a_1+c_2\leqslant d_2=r_2+p_2\\ \vdots\\ a_n=a_{n-1}+c_n=r_1+c_1+c_2+\cdots+c_n\leqslant d_n=r_n+p_n\\ r_1+\sum_{i=1}^{n}c_i\leqslant r_n+p_n\end{gathered}$$

2) 当 $a_{i-1}\leqslant r_i$，$i=2,3,\cdots,n$ 时，则有：

$$\begin{gathered}a_1=r_1+c_1\leqslant d_1=r_1+p_1\\ a_2=r_2+c_2\leqslant d_2=r_2+p_2\\ \vdots\\ a_n=r_n+c_n\leqslant d_n=r_n+p_n\end{gathered}$$

则当 $c_i\leqslant p_i$ 即可被成功调度。

### 3.2 算法思路

1) 计算该级别下每个job的关键因子，并根据关键因子的大小决定优先级高低进行优先级排序，并根据定理判定是否可以调度在同一个核上。

2) 检查是否有关键级别转化(关键级别转化是指系统在运行时关键级别由高到低或由低到高的转化过程)，如果有则根据已经转化的当前级别计算每个job的关键因子，并在此决定新的优先级顺序，并与转化前的优先级顺序相比较，记录有优先级推后的job以及该job优先级未推后时后面的job顺序。

3) 针对未推后时的job后面的调度顺序，根据定理判定这些job是否能够调度在同一个处理器上。如果可以，根据这些job的释放时间和各个级别下的最坏执行时间计算最早执行窗口 $W_i^e$ (正向时间分割函数)和最迟执行时间窗口 $W_i^l$ (反向时间分割函数)，并求出它们的公共空闲时间窗口。在公共空闲窗口时间调度推后的job，如果不可以，则只能另外分配处理器。

4) 如果没有关键级别转化，或者关键级别转化后没有优先级推后的job，则继续按照该调度顺序执行。

### 3.3 算法实例

根据文献[2, 10]，研究一个2个关键级别的系统，job集合为 $J=\{J_1,J_2,J_3,J_4\}\}$，其中，$J_2$ 的关键级别为1，其他3个的关键级别均为2。其中，$J_i=(r_i,d_i,x_i,[c_i(1),c_i(2)])$，$i=1,2,3,4$。

$J_1$=(0,3,2,[1,2])；　　$J_2$=(0,3,1,[2,2])

$J_3$=(0,5,2,[1,1])；　　$J_4$=(3,5,2,[1,2])

可以计算出它们的关键因子以及空闲时间窗口，如表1所示。

**表1　各个job的关键因子与时间窗口计算结果**

| 关键级别 | 关键因子与时间窗口 | $J_1$ | $J_2$ | $J_3$ | $J_4$ |
|---|---|---|---|---|---|
| 1 | 关键因子 | 0.095 | 0.095 | 0.057 | 0.057 |
| | 最早执行时间窗口 | [0,1] | [0,2] | [0,1] | [3,4] |
| | 最迟执行时间窗口 | [2,3] | [1,3] | [4,5] | [4,5] |
| | 空闲时间窗口 | [1,2] | 无 | [1,4] | 无 |
| 2 | 关键因子 | 0.19 | 0.048 | 0.057 | 0.114 |
| | 最早执行时间窗口 | [0,2] | [0,2] | [0,1] | [3,5] |
| | 最迟执行时间窗口 | [1,3] | [1,3] | [4,5] | [3,5] |
| | 空闲时间窗口 | 无 | 无 | [1,3] | 无 |

从表中可以看出，当关键级别为1时，$J_1$与$J_2$关键因子相同，优先级别相同，优先于$J_3$和$J_4$，而$J_3$与$J_4$优先级别相同则可以选择$J_1,J_2,J_3,J_4$为调度顺序，且根据定理可以判断出这种顺序可以满足在同一个核上的调度条件。

当转化为关键级别为2时，优先顺序为$J_1,J_4,J_3,J_2$。通过定理可以判定该顺序中前3个job可以调度在同一个核上，而$J_2$无法与这3个调度在同一个核上，$J_2$需要被分配在其他核上运行。

# 4　性能比较

## 4.1　仿真实验方法

仿真实验采用文献[9]中方式，随机产生1 000个任务集，每个任务集包含$n$(3～10之间)个相互独立的任务且带有总共CPU利用率为0.8。使用调和函数和非调和函数任务集，通过输入值控制每个任务的实际参数。

对于调和函数任务集，一个任务$2^k$×1 000的周期，这里$k$是一个按惯例随机分布在整型值，在0～5之间。

对于非调和任务集，一个任务的周期是$k$×2 000，这里$k$是一个正常的分布式随机整型值，在1～16之间。与文献[9]所不同的是每个任务$T_i$是一个3个级别的任务。最坏执行时间和相对级别的利用率因子$u_i^k(k=1,2,3$相关联$)$，从方程$c_i(k)=u_i^k p_i$中获得，且$\sum_{i=1}^{n}u_i^k=0.8$。仿真实验中还针对不同的任务集，采用随机输入了6个级别转化时间值$t=\{t_1,t_2,t_3,t_4,t_5,t_6\}$　且　$50\leqslant t_{i-1}<t_i\leqslant 2\ 000$　，$i=1,2,\cdots,6$，用以指示级别从低到高转化的时刻，以及从高到低级别转化的时刻。

## 4.2　实验结果分析

仿真实验将CAPA算法、OCBP算法以及本文所提出的FBTWP-CFP算法进行了比较。

**表2　两种核数情况下平均丢失死限率比较**

| | 双核 | | 四核 | |
|---|---|---|---|---|
| | 调和函数 | 非调和函数 | 调和函数 | 非调和函数 |
| CAPA | 0.134 | 0.121 | 0.065 7 | 0.057 2 |
| OCBP | 0.145 | 0.141 | 0.081 1 | 0.072 1 |
| FBTWP-CFP | 0.088 | 0.071 | 0.043 9 | 0.041 5 |

首先针对0.8利用率的100个任务集，每个任务集中10个任务在双核和四核情况下10个公共周期内对任务丢失死限率进行了统计，任务丢失死限率是指丢失死限任务数量与总任务数量的比值，结果如表2所示。FBTWP-CFP在整个过程中丢失死限数量比其他两种算法要好些。非调和函数死限丢失率比调和函数丢失率较低，原因是有些非调和函数的周期长一些。

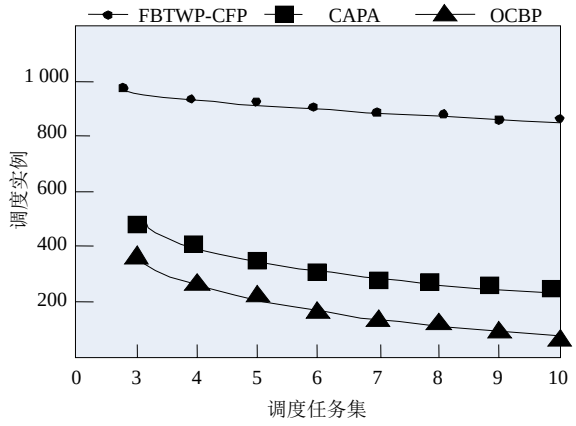


图1　调和函数任务集

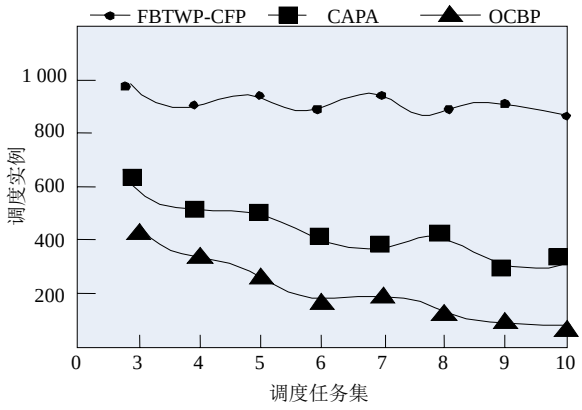


图2　非调和函数任务集

其次是在四核数量情况下，在死限之前完成的任务数量的对比，从图1和图2可以看出无论是调和函数还是非调和函数，FBTWP-CFP完成的任务数量远远超过了其他两种算法。

(下转第286页)

amplifier[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2007, 54(12): 1042-1046.
[4] DUONG Q H, NGUYEN T K, LEE S G. Ultra low-voltage low-power exponential voltage-mode circuit with tunable output range[C]//Proceedings of the International Symposium on Circuits and Systems. [S.l.]: IEEE, 2004.
[5] LEI Qian-qian, CHEN Zhi-ming, SHI Yin, et al. A low-power CMOS VGA with 60 dB linearly controlled gain range for GPS application[C]//9th International Conference on Solid-State and Integrated-Circuit Technology. Beijing: IEEE Press, 2008: 1669-1672.
[6] KALENTERIDIS V, VLASSIS S, SISKOS S. A CMOS linear-in-dB VGA based on exponential current generator [C]//2011 6th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS). Athens: IEEE, 2011.
[7] LUIS H, PIMENTA, MORENO R L. An ultra low-voltage ultra low-power CMOS threshold voltage reference[J]. IEICE Transactions on Electronics, 2007, E90-c(10): 2044-2050.
[8] 余国义. 低压低功耗CMOS基准参考源的设计[D]. 武汉: 华中科技大学, 2006.
YU Guo-yi. Design of low-power CMOS reference[D]. Wuhan: Huazhong University of Science and Technology, 2006.
[9] LEE H, MOHAMMADI S. A 3 GHz subthreshold CMOS low noise amplifier[C]//IEEE RFIC Symp. San Francico, CA: IEEE Press, 2006: 494-497.
[10] LEE H, MOHAMMADI S. Subthreshold low phase noise CMOS LC VCO for ultra low power applications[J]. IEEE Microwave and Wireless Components Letters, 2007, 17(11): 796-798.
[11] 杜永乾, 庄奕琪, 李小明, 等. 一种新型无源UHF RFID带隙基准电路[J]. 西安电子科技大学学报(自然科学版), 2013, 40(2): 148-152.
DU Yong-qian, ZHUANG Yi-qi, LI Xiao-ming, et al. A new type bandgap reference for UHF RFID tag[J]. Journal of Xidain University(Natural Science Edition), 2013,40(2): 148-152.
[12] GIUSTOLISI G, PALUMBO G, CRISCIONE M, et al. A new voltage reference topology based on subthreshold MOSFETs[C]//Proceedings of the 28th European Conference on Solid-State Circuits. Florence, Italy: IEEE, 2002.
[13] LIU W, JIN X, CHEN J, et al. BSIM3v3.2.2 MOSFET model users'manual[EB/OL]. [2009-06-01]. http://www.eecs.berkeley.edu/Pubs/TechRpts/1999/ERL-99-18.pdf.
[14] De LA CRUZ BLAS C A, LÓPEZ-MARTÍN A. Novel low-power high-dB range CMOS pseudo-exponential cells[J]. ETRI Journal, 2006, 28(6): 732-738.
[15] MAUNDY B, GIFT S. Novel pseudo-exponential circuits[J]. IEEE Transactions on Circuits and Systems, 2005, 52(10): 675-679.

编 辑 张 俊

---

(上接第271页)

## 参 考 文 献

[1] DE NIZ D, LAKSHMANAN K; RAJKUMAR R. On the scheduling of mixed-criticality real-time task sets[C]// Proceedings of the 30th Real-Time Systems Symposium. Washington: IEEE Computer Society, 2009: 291-300.
[2] BARUAH S, LI Hao-han, STOUGIE L. Towards the design of certifiable mixed-criticality systems[C]//Proceedings of the 16th Real-Time and Embedded Technology and Applications Symposium (RTAS). Stockholm, Sweden: IEEE Computer Society, 2010: 13-22.
[3] LI Hao-han, BARUAH S. Load-based schedulability analysis of certifiable mixed-criticality systems[C]// Proceedings of the 10th ACM International Conference on Embedded Software. Scottsdale, Arizona, USA: ACM, 2010: 99-108.
[4] BARUAH S, BONIFACI V, LI Hao-han. Scheduling real-time mixed-criticality jobs[J]. Mathematical Found-Ations of Computer Science, 2010, 62(81): 90-101.
[5] BARUAH S, BONIFACI V, GIANLORENZO D A, et al. Mixed-criticality scheduling of sporadic task systems algorithms[C]//Proceedings of the 19th Annual European Symposium. Saarbrücken, Germany: Springer-Verlag, 2011, 9: 5-9.
[6] HAOHAN L, BARUAH S. An algorithm for scheduling certifiable mixed-criticality sporadic task systems[C]// Proceedings of the 31st Real-Time Systems Symposium (RTSS). California, USA: IEEE Computer Society, 2010: 183-192.
[7] SHA L, RAJKUMAR R, LEHOCZKY J P. Priority inheritance protocols: an approach to real-time synchronization[J]. IEEE Transactions on Computers, 1990, 39(19): 1175-1185.
[8] MOLLISON M S, MOLLISON J P, ERICKSON J H, et al. Mixed-criticality real-time scheduling for multicore systems[C]//IEEE 10th International Conference on Computer and Information Technology (CIT). Bradford, UK: IEEE Computer Society, 2010: 1864-1871.
[9] HUANG Ming-huang, GILL C, LU Cen-yang. Implementation and evaluation of mixed-criticality scheduling approaches for periodic tasks[C]//Proceeding of the 18th Real-Time and Embedded Technology and Applications Symposium (RTAS). Beijing: IEEE Computer Society, 2012: 23-32.
[10] BARUAH S. Mixed criticality schedulability analysis is highly intractable[EB/OL]. [2009-08-25]. http://www.cs.unc.edu/~baruah/Pubs.shtml.

编 辑 张 俊