2008 年 8 月 第 45 卷第 4 期 四川大学学报(自然科学版) Journal of Sichuan University (Natural Science Edition) Aug. 2008 Vol. 45 No. 4

文章编号: 0490-6756(2008)04-0779-06

# 一种支持多级关键度任务的容错实时调度算法

曾 坤, 黎忠文
(厦门大学信息科学与技术学院, 厦门 361005)

**摘 要**: 针对当前对支持多关键度的实时系统没有涉及容错功能的研究情况,本文提出了支持多级关键度任务的容错实时系统模型,通过对模型中任务出错后关键度任务集合的响应时间分析,提出了新的算法,我们称之为补任务启动及容错优先级确定算法,算法假设在运行该算法的时刻开始系统不会再次出错,基于这个假设计算每个任务的响应时间,从而决定要不要启动出错任务的补任务以及容错优先级如何分配. 该算法在保证系统的容错能力的同时提高了任务的完成率,使系统吞吐量损失最小,从而提高系统的可靠性. 最后经过实例对该算法进行验证.

**关键词**: 容错实时系统;多级关键度;容错优先级;响应时间
**中图分类号**: TP316 **文献标识码**: A

# A fault-tolerant real-time scheduling algorithm supporting tasks of multilevel criticality

*ZENG Kun, LI Zhong-Wen*
(School of Information Science and Technology, Xiamen University, Xiamen 361005, China)

**Abstract**: Based on the current research area of the real-time system of supporting tasks of multilevel criticality not including fault-tolerance technology, a fault-tolerant real-time system model supporting tasks of multilevel criticality is proposed. With the worst-case response time schedulability analysis when an fault happen, a new algorithm that decide whether to startup the alternative task and how to assign the fault-tolerant priority is proposed too. Not only does this algorithm enable the system to recover errors, it also keep the system's throughout as much as possible. The effectiveness of the approach is evaluated by an instance.

**Key words**: fault-tolerant real-time system, multilevel criticality, fault-tolerant priority, response time

## 1 引 言

在后 PC 时代,计算机技术在安全关键(SCS)系统中得到了广泛的应用,而系统的可信性问题却越来越突出. 现在对系统可信性的研究大都是基于子系统的关键度进行的,通过一系列措施使关键程度较低的子系统不能影响关键度较高的子系统,从而提高系统的可信性. 文献[1~6]对关键度进行了深入的研究. 文献[4]通过比较 Biba 完整性控制规则[7]以及 Clark &Wilson 完整性控制规则[8],提出

收稿日期: 2007-01-24; 修回日期: 2007-03-09
基金项目: 广东省自然科学基金(06029667);厦门大学院士基金(0630-E23011);厦门大学新世纪优秀人才基金(0000-X07116);福建省自然科学基金(2008J0034)
作者简介: 曾坤(1983-),男,福建漳州人,硕士研究生,主要研究领域为计算机网络,容错以及实时系统.
通讯作者: 黎忠文. E-mail: lizw@xmu.edu.cn

了新的面向对象的完整性控制规则,用以保证共享低关键度子系统的缺陷不会危及高关键度子系统的可信性. 文献[2]介绍了在 GUARD 项目中提出一种基于多关键度模型和面向对象框架的完整性控制规则以及应用于该规则正则描述和确认的模型检验技术. 文献[3]定量描述了实时任务的关键度,定义关键度是通过任务允许错过时限的次数来描述任务的"软硬"程度,并提出了相应的一套实时调度算法,从而提高了 SCS 系统的防危性. 文献[4]提出了最大关键度优先的调度算法,这是一种混合型的优先级实时调度算法. 该算法充分利用处理器资源,并且在发生瞬时过载时保证关键任务不受非关键任务的影响,增强了实时系统的防危性. 文献[5]研究了不同关键度子系统相互作用应遵守的防危访问控制规则,防止了不同关键度子系统间的有害干扰. 文献[6]提出了一种支持多级关键度子系统共享同一系统资源的集成式高可信保障体系结构以及基于两级结构化调度方法的时间隔离机制.

容错作为一种通用的可信性保障机制,目的是使系统在运行出现错误时能够继续提供标准或降级服务. 因此容错技术的研究也是当今研究一大热点,而大部分的容错技术的研究都是基于硬实时系统. 在硬实时系统中,容错调度算法一般有 3 个模型[9]:硬件容错模型,非精确计算模型以及软件容错模型. 文献[9～13]都基于软件容错模型对容错硬实时调度进行研究. 文献[9]新提出了两种类似 EDF 的动态实时调度算法,提高了任务的完成率. 文献[10]也提出一种容错调度算法,这种算法对于容错优先级采用继承任务在正常情况下的优先级的分配策略. 文献[11]提出了相对灵活的容错优先级分配策略,允许提高任务的容错优先级从而提高系统的容错能力. 文献[12]针对基本算法一个任务运行出错可能导致一系列任务由于不能在实现前完成而出错的缺点,对其进行改进,提出了新的算法,避免了错误的"蔓延". 文献[13]在[11]的基础上提出了降低任务容错优先级的分配策略. 当容错优先级在继承和提高的分配策略下不能提高容错能力的情况下,这种新的分配策略能够提高系统的容错能力.

为了更好的保证系统安全性和防危性,我们在文献[14,15]提出了新的体系结构 PASD,利用了防危核(壳)以及关键度完整性机制实现系统的 S&S(Security & Safety)服务,并且使用反射技术实现了完整性机制的规则. 在文献[16]中,我们采用检测点时间冗余容错和优先级提升思想提出了响应时间分析方法,为基于防危核(壳)的 SCS 系统提供了离线可调度分析工具,并为系统设计了一种基于检测点的容错方案.

在我们的研究过程中发现,基于硬实时系统的调度算法[17,18]并不适用于支持多关键度任务的实时系统,同时当前对支持关键度的实时系统的研究并不涉及系统的软件容错能力,对容错调度算法的研究也都基于硬实时系统. 针对这种情况,我们结合文献[11～13]通过对容错实时任务基于响应时间的可调度性分析,提出适用于支持多关键度的实时系统的容错调度算法. 该调度算法能够在任务出错后确定要不要启动出错任务的补任务以及容错优先级,充分利用处理器资源,避免处理器做无用功,并且使尽可能多的任务在其时限前完成.

## 2 支持多级关键度任务的容错实时系统

### 2.1 支持多级关键度任务的容错实时系统模型

定义一个实时任务集合 $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$, $\tau_i$ 表示任务,是在没有发生故障时系统调度对象. 任务 $\tau_i$ 定义为一个五元组 $< C_i, D_i, T_i, \phi_i, \overline{C_i} >$, 其中, $C_i$ 表示任务的执行时间; $D_i$ 表示任务的时限; $T_i$ 表示周期任务的周期; $\phi_i$ 表示任务的关键度[4],我们在 2.2 节对其进行详细地介绍; $\overline{C_i}$ 表示对应补任务的执行时间. 对非周期任务而言, $T_i$ 表示任务到来的时间间隔最小值. 我们用 $\overline{p_i}$ 表示 $\tau_i$ 的容错优先级,即 $\overline{\tau_i}$ 的优先级. 研究的前提包括:

(1) $D_i = T_i$.

(2) 如果 $\tau_i$ 和 $\overline{\tau_j}$ 的优先级相同并且都处于就绪状态,则 $\overline{\tau_j}$ 优先执行.

(3) 运行补任务 $\overline{\tau_i}$ 可以是对任务 $\tau_i$ 的重新运行,也可以是运行任务 $\tau_i$ 的替代部分[9].

(4) 当任务 $\tau_i$ 运行出错时,系统通过响应时间分析决定是否要调用补任务 $\overline{\tau_i}$. 而当补任务 $\overline{\tau_i}$ 运行出错时,所采取的容错技术还是通过响应时间分析决定是否再次执行 $\overline{\tau_i}$.

### 2.2 关键度

关键度是描述实时任务紧急程度的变量. 通常,关键度越高的任务,允许其错过时限的次数越少. 为了定量描述关键度这个概念,文献[3]给出了以下定义:假设任务 $\tau$ 被连续调用 $m$ 次,如果至

少有 $n$ 次满足其时限,则任务 $\tau$ 的关键度 $\phi$ 为 $[n, m]$,即 $\phi = [n, m]$,其中 $m$ 为描述任务关键度的时间的范围限制窗口.

基于关键度的定义,当任务被调度时将处于以下 3 种状态:

(1) 正常状态:即使任务在下次运行过程中错过时限,但仍然可以满足其关键度要求.

(2) 紧急状态:如果任务在下次运行过程中错过时限,则不能满足其关键度要求.

(3) 失效状态:任务已不能满足其关键度要求. 任务处于以上 3 种状态中的一种,为了达到 SCS 系统的要求,我们必须让多级关键度任务尽可能处在正常状态,偶尔处于紧急状态,绝对不能处于失效状态. 为了对这些任务进行灵活的调度,系统的任务优先级就不能是恒定不变的. 当任务即将进入紧急状态,应尽可能的让该任务下次运行不错过时限. 如果任务已经进入紧急状态,应尽可能的立即调度该任务,并保证该任务在本次运行不错过时限. 距离失效状态越近的任务应该优先运行,其分配的优先级越高,反之则越低.

设任务 $\tau_i$ 的关键度 $\phi_i = [n_i, m_i]$,$\delta_j$ 为任务 $\tau_i$ 的当前最近状态,即第 $j$ 次运行时所处的状态,则任务 $\tau_i$ 的前 $m_i$ 次连续调用实例组成的状态集合为 $s_i = \{\delta_{j-m+1}, \cdots, \delta_{j-1}, \delta_j\}$. 另外设 $l_j(n_i, s_i)$ 为从右到左第一次满足 $n_i$ 个 1 的位置,根据 SUDF 算法[8],则任务 $\tau_i$ 在下一次运行时所分配的优先级 $p_i^{j+1}$ 为:

$$p_i^{j+1} = \begin{cases} m_i - l_j(n_i, s_i) + 1, & l_j(n_i, s_i) \leq m_i \\ 0, & l_j(n_i, s_i) > m_i \end{cases} \tag{1}$$

## 3　响应时间分析

在进行最坏响应时间分析之前,先给出以下定义:

$$hpe(i) = \{\tau_j \in \Gamma \mid p_j \geq p_i\}$$

$$spe(i) = \{\tau_j \in \Gamma \mid p_j \geq \overline{p}_i\}$$

$$\delta(i, j) = \begin{cases} 0, & p_i > \overline{p}_j \\ 1, & p_i \leq \overline{p}_j \end{cases}$$

$$\lambda(i, j) = \begin{cases} 0, & \underline{p}_i > \overline{p}_j \\ 1, & \underline{p}_i \leq \overline{p}_j \end{cases}$$

$$\gamma(i, j) = \begin{cases} 0, & p_i < p_j \\ 1, & p_i \geq p_j \end{cases}$$

$$\text{rectify}(k) = \begin{cases} 0, & k \leq 0 \\ k - 1, & k > 0 \end{cases}$$

$$\text{time}(i, j, k) = \begin{cases} \min\{p_j - p_i, k\}, & m_i - n_i + 1 \geq p_j \\ k, & m_i - n_i + 1 < p_j \end{cases}$$

$R_i$ 表示的是假设从当前时刻开始不会有任务出现故障的情况下的任务 $\tau_i$ 的响应时间. 令 $t_{hl}$ 为任务该次运行从准备就绪开始计时到最近一次错误发生的时间,$t_{hr}$ 为从最近一次错误发生的时刻开始计时到现在的时间以及 $t_h$ 为从最近一次错误发生开始计时到现在补任务的有效执行时间,这里所说的错误指的是通过执行补任务可以消除的错误. 如果任务在该次运行中至今还没有发生错误,$t_{hr}$ 就表示任务该次运行从准备就绪开始计时到现在的时间,$t_{hr}$ 和 $t_h$ 都等于 0. 这 3 个量都是实时更新的,每次实时系统中的周期性任务就绪时,就把该任务的这 3 个变量重新设置为 0;如果任务发生错误,就更新这两个变量,把 $t_{hl}$ 设置为原来的 $t_{hl}$ 和 $t_{hr}$ 之和,然后把 $t_{hr}$ 和 $t_h$ 设置为 0. 我们可以为每个任务分配 3 个计时器,存放这 3 个变量.

另外定义两个全局数组 fault 和 error. 当前时刻任务 $\tau_i$ 发生错误但是其补任务还没有运行完毕时 fault[i] = 1. fault 数组初始化为 0,一旦某个任务出现错误并且系统调度该任务的补任务时,就把 fault 数组该任务的对应位设置为 1,当补任务在时限前完成或者任务时限到来时重新设置为 0. 当前时刻任务 $\tau_i$ 的该次运行被杀死时 error[i] = 1. error 数组也初始化为 0,当任务被杀死时对应位被设置为 1,当任务再次就绪时又重新把对应位设置为 0.

然后我们先考虑一种特殊情况:当出错任务的当前优先级等于 1 时,说明该任务紧急程度最高,如果任务的这次运行不能在时限前完成就会导致严重的系统错误,因此其补任务的紧急程度也必须最高. 所以一旦优先级为 1 的任务出错,系统启动其补任务,并且设置其容错优先级为 1.

当优先级不为 1 的任务出错时,算法将根据各个任务的相应时间和时限的关系来确定是否启动其补任务以及容错优先级该该怎么分配. 各个任务的最坏响应时间如式(2) 所示.

$$R_i = \begin{cases} \sum_{\tau_j \in spe(\tau_i)-\{\tau_i\}} \text{time}\left(j, i, \text{rectify}\left(\left|\frac{R_i - (t_{hl})_i - (t_{hr})_i - (D_j - (t_{hl})_j - (t_{hr})_j)}{T_j}\right|\right)\right) C_j + \\ \quad \sum_{\text{error}[l]=0} \text{fault}[l] * \lambda(i,l)(\overline{C}_l - (t_h)_l) + (1 - \text{fault}[l])(1 - \delta(l,i))(C_l - (t_h)_l) + \\ \quad (t_{hl})_i + (t_{hr})_i, \qquad \text{fault}[i] = 1 \\ \sum_{\tau_j \in hpe(\tau_i)-\{\tau_i\}} \text{time}\left(j, i, \text{rectify}\left(\left|\frac{R_i - (t_{hl})_i - (D_j - (t_{hl})_j - (t_{hr})_j)}{T_j}\right|\right)\right) C_j + \\ \quad \sum_{\text{error}[l]=0} \text{fault}[l] * \delta(i,l)(\overline{C}_l - (t_h)_l) + (1 - \text{fault}[l])\gamma(l,i)(C_l - (t_h)_l) + \\ \quad (t_{hl})_i, \qquad \text{fault}[i] = 0 \wedge \text{error}[i] = 0 \end{cases} \tag{2}$$

优先级比 $\overline{\tau}_i$ 高的任务 $\tau_j$ 下次就绪时刻是距当前 $D_j - (t_{hl})_j - (t_{hr})_j$ 的时间,从 $\tau_j$ 下次就绪到 $\overline{\tau}_i$ 该次运行在其时限前完成这段时间总共运行

$$\left|\frac{R_i - (t_{hl})_i - (t_{hr})_i - (D_j - (t_{hl})_j - (t_{hr})_j)}{T_j}\right|$$

次,但根据 SUDF 算法及式(1),$\tau_j$ 的每次成功运行都可能使 $\tau_j$ 优先级降低,因此在 $\tau_j$ 次运行中能够抢占 $\tau_i$ 的次数为

$$\sum_{\tau_j \in spe(\tau_i)-\{\tau_i\}} \text{time}(j, i, \text{rectify} \cdot \left(\left|\frac{R_i - (t_{hl})_i - (t_{hr})_i - (D_j - (t_{hl})_j - (t_{hr})_j)}{T_j}\right|\right)) C_j.$$

再者,在当前时刻处于就绪状态的任务只要优先级比 $\overline{\tau}_i$ 高并且没有被杀死就可以抢占 $\overline{\tau}_i$,这部分的时间为 $\sum_{\text{error}[l]=0 \wedge l \neq i} \text{fault}[l] * \lambda(i,l)(\overline{C}_l - (t_h)_l) + (1 - \text{fault}[l])(1 - \delta(l,i))(C_l - (t_h)_l)$. 最后 $R_i$ 还包括 $\tau_i$ 从就绪到最近一次发生故障前的时间 $(t_{hl})_i + (t_{hr})_i$ 以及 $\overline{\tau}_i$ 的运行时间 $\overline{C}_i$. 对于那些在此次运行中没有出现故障的任务 $\tau_i$,求其相应时间 $R_i$ 和求出错任务的相应时间的方法类似.

综合以上两种情况就可以得到计算任务最坏响应时间的式(2),取 $\Gamma_d = \{\tau_i \mid R_i > D_i\}$. 当任务 $\tau_k$ 发生错误时,有:

(1) 我们首先试着分配容错优先级为其原任务的优先级即 $p_{\overline{\tau}_k} = p_{\tau_k}$. 如果 $\Gamma_d = \phi$,说明所有的任务都可以在其时限前完成,则任务集合同样是可调度的. 如果 $\tau_k \in \Gamma_d$,转(2);如果 $\tau_k \notin \Gamma_d, \tau_i \in \Gamma_d \wedge i \neq k$,转(3).

(2) $\tau_k \in \Gamma_d$ 说明补任务 $\overline{\tau}_k$ 不能在时限之前完成,我们可以通过提高 $\overline{\tau}_k$ 的优先级,减少 $R_i$. 但是我们必须防止这样一种情况发生:$\tau_i \in \Gamma_d \wedge p_i \geq p_k$ 或者 $\overline{\tau}_i \in \Gamma_d \wedge \overline{p}_i \geq p_k$ 即存在某个优先级比 $\tau_k$ 高的任务(或补任务) 由于 $\overline{\tau}_k$ 的运行而使其在时限到来之前不能执行完毕. 如果系统出现上述情况,启动补任务 $\overline{\tau}_k$ 是得不偿失的,这时我们采取的策略是不启动补任务.

(3) $\tau_k \notin \Gamma_d, \tau_i \in \Gamma_d \wedge i \neq k$,则 $\tau_i$ 的优先级肯定比 $\tau_k$ 低,因为比 $\tau_k$ 优先级高的任务 $\tau_i$ 在 $\tau_k$ 发生错误前跟发生错误后的 $R_i$ 都是按照式(2) 求得,$\tau_k$ 发生错误前 $R_i \leq D_i$ 成立,$\tau_k$ 发生错误后的 $R_i \leq D_i$ 也成立. 因此我们可以试着降低 $\overline{\tau}_k$ 的优先级,从而在保证 $\overline{\tau}_k$ 在时限之前完成的前提下让尽可能多的任务能够在时限之前完成.

如果在上述计算补任务优先级的过程中,找到了数值 $p$,当 $\overline{p}_k = p$,使得 $\Gamma_d = \emptyset$ 成立,则 $p$ 就是任务 $\overline{\tau}_k$ 优先级的一个赋值. 但是如果不论任务 $\overline{\tau}_k$ 优先级为多大时 $\Gamma_d \neq \emptyset$,这时,我们就要找数值 $p$,当 $\overline{p}_k = p$ 时,$\Gamma_d \wedge hpe(k) = \emptyset$ 成立且 $T_d = \sum_{\tau_i \in \Gamma_d} C_i$ 的值最小.

**定理 1** 在任务 $\tau_j$ 的优先级 $p_j > p_i \wedge p_j > \overline{p}_i$ 条件下,出错任务 $\tau_i$ 出错前计算的响应时间 $R_j$ 跟出错后计算的响应时间 $R_j'$ 相等,即 $R_j = R_j'$

**证明** 因为 $p_j > p_i \wedge p_j > \overline{p}_i$,所以任务 $\tau_j$ 比 $\tau_i$ 的优先级高,而且比 $\overline{\tau}_i$ 的优先级高. 不管是 $\tau_i$ 还是 $\overline{\tau}_i$ 运行都不能抢占 $\tau_j$,因此 $\tau_i$ 的运行状况不会影响到 $\tau_j$,所以出错任务 $\tau_i$ 出错前计算的响应时间跟出错后计算的响应时间相等.

**定理 2** 在任务 $\tau_j$ 的优先级 $p_j < p_i \wedge p_j < \overline{p}_i \wedge C_i = \overline{C}_i$ 条件下,出错任务 $\tau_i$ 出错前计算的响应时间 $R_j$ 跟出错后计算的响应时间 $R_j'$ 满足一下关系:$R_j' - R_j \geq (t_h)_i$

**证明** 因为 $p_j < p_i \wedge p_j < \overline{p}_i$,所以任务 $\tau_j$ 比 $\tau_i$ 和 $\overline{\tau}_i$ 的优先级都低. 不管是 $\tau_i$ 还是 $\overline{\tau}_i$ 运行都会抢占 $\tau_j$,因此 $\tau_i$ 出错以前的该次运行对 $\tau_i$ 的运行是没有贡献的,$(t_h)_i$ 的这段时间处理器做的是无用

功.因此出错任务 $\tau_i$ 出错前计算的响应时间 $R_j$ 跟出错后计算的响应时间 $R_j'$ 至少相差 $(t_h)_i$,因此 $R_j' - R_j \geq (t_h)_i$.

## 4　实例分析

由于目前我们尚没有查到针对支持多关键度容错调度算法的资料,不能与本文的实验结果相比较.为了说明算法能够比较好的实现容错功能,假设存在如表 1 所示的任务,当 $t = 0$ 时 3 个任务同时就绪.

表 1　各个实时任务当前的参数

Tab. 1　Current parameters of each real-time task

| | $C_i$ | $D_i$ | $T_i$ | $\phi_i$ | $p_i$ |
|---|---|---|---|---|---|
| $\tau_1$ | 20 | 100 | 100 | [1,2] | 2 |
| $\tau_2$ | 60 | 150 | 150 | [3,5] | 3 |
| $\tau_3$ | 60 | 300 | 300 | [3,6] | 4 |

其中 $p_i$ 表示任务的当前优先级.图 1 表示的是系统没有任务发生故障时的调度情况.图中每个小格代表的时间为 20 s.

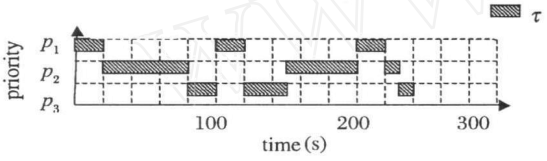


图 1　无错误发生时的任务调度

Fig. 1　Tasks scheduling when no error occur

假设当 $t = 100$ s 时 $\tau_3$ 出现故障时,系统采用补任务启动即容错优先级确定算法得到当 $\overline{p_3} = p_3$,算得 $R_1 = 20 < 100$, $R_2 = 80 < 150$, $R_3 = 260 < 300$,说明该进程组是可调度的.其系统调度情况如图 2 所示.

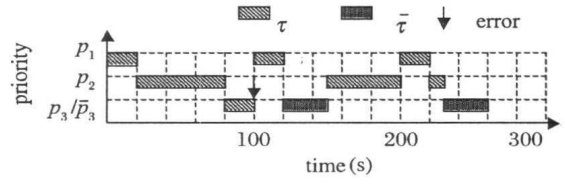


图 2　当 $t = 100$ s 时 $\tau_3$ 错误发生的进程调度

Fig. 2　Tasks scheduling when there comes a error when $t = 100$ s

假设当 $t = 230$ s 时 $\tau_3$ 也出现故障时,系统采用补任务启动即容错优先级确定算法重新计算,不管 $\overline{p_3}$ 取什么值, $T_d$ 都不为 0.当 $\overline{p_3} = p_3$ 时 $R_1 = 20 < 100$, $R_2 = 140 < 150$, $R_3 = 320 > 300$, $T_d$ 取最小值 $T_d = 60$, $\Gamma_d = \{\overline{\tau_3}\}$,算法就杀死进程 $\overline{\tau_3}$,启动进程 $\overline{\tau_2}$.其调度情况如图 3 所示.

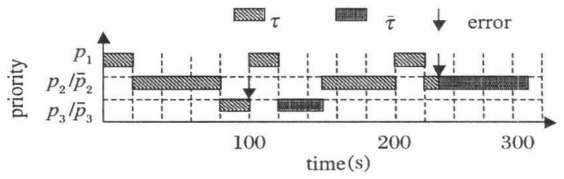


图 3　当 $t = 230$ s 时又有 $\tau_2$ 错误发生的进程调度

Fig. 3　Tasks scheduling when there comes a error when $t = 230$ s as well

通过上述例子可以看出,补任务启动即容错优先级确定算法可以使系统在保证容错能力的同时,通过使 $T_d$ 的值最小达到系统的吞吐量尽量损失最小的目的.

## 5　结　语

在支持多级关键度任务的实时容错系统中,当某个任务出现故障时,系统必须决定是否启用该补任务以及怎么确定容错优先级,保证系统的容错能力以及吞吐量.为了解决这个问题,本文基于响应时间分析提出了一种新的算法——补任务启动即容错优先级确定算法,经过实例证明:该算法保证在保证系统的容错能力的同时提高了任务的完成率,使系统吞吐量损失最小,从而提高系统的可靠性.

**参考文献**:

[1] Totel E, Blanquart J P, Deswarte Y, *et al*. Supporting multiple levels of criticality[C]// Fault-Tolerant Computing, 1998. Digest of Papers, Twenty-Eighth Annual International Symposium:70

[2] Fantechi A, Gnesi S, Semini L. Formal formal description and validation for an integrity policy supporting multiple level of criticality[C]// Proceedings of the conference on Dependable computing for critical applications. Washington. DC, USA: IEEE Computer Society, 1999: 129.

[3] 杨仕平,黄耕文,刘校矢. 可支持多级关键度任务的实时操作系统[J]. 系统工程与电子技术, 2006, 28(4): 615.

[4] 杨仕平,桑楠,吴新勇,等. 高可信赖实时操作系统的防危调度机制[J]. 电子科技大学学报, 2006, 35

(1):111.

[5] 杨仕平,桑楠,吴新勇,等. 基于多级关键度的高可信安全关键系统[J]. 系统工程与电子技术,2004,26(2):277.

[6] 杨仕平,熊光泽,桑楠. 安全关键实时系统的高可信集成技术的研究[J]. 电子学报,2003,31(8):1237.

[7] Biba K J. Integrity considerations for secure computer systems[R]// Technical Report ESD-TR-76-372, USAF Electronic Systems Division, Hanscom Air Force Base, Bedford, Massachusetts, April 1977.

[8] Clark D D, Wilson D R. A comparison of commercial and military computer security policies [C]// IEEE sumposium on security and privacy. New York: IEEE Computer Society Press, 1987.

[9] 李庆华,韩建军,Abbas Essa,等. 硬实时系统中基于软件容错的动态调度算法[J]. 软件学报,2005,16(1):101.

[10] Punnekkat S. Scheduling analysis for fault tolerant real-time systems[D]. Hesingto York, UK: University of York, 1997.

[11] De A. Lima George M, Burns Alan. An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems[J]. IEEE Transactions on Computers, 2003, 52(10):1332.

[12] Han C C, Shin K G, Wu J. A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults[J]. IEEE Transactions on Computers, 2003, 52(3):362.

[13] 李俊,阳富民,卢炎生. 一种可行的容错实时系统可调度性分析[J]. 软件学报,2005,16(8):1513.

[14] Li Z W, Qiu Z P. A new type of security and safety architecture for distributed system: models and implementation[C]// Models and Implementation, Proceeding of the Third International Conference on Information Security. New York, NY, USA: ACM, 2004: 107.

[15] Li Z W. Security and safety assurance architecture: models and implementation (supporting multiple levels of criticality) [C]// 6th International conference on algorithms and architecture for parallel processing, (ICA3PP), Lecture notes in computer science (LNCS). [s.l.], Springer Verlag, 2005.

[16] 黎忠文,程亮,熊光泽. 基于防危核(壳)的安全关键硬实时系统响应时间的分析[J]. 电子学报,2006,34(4):58.

[17] 穆阿里,吴仲光,张昭瑜,等. 一种多特征综合地实时调度算法[J]. 四川大学学报:自然科学版,2005,42(3):621.

[18] 徐文清,杨红雨. 一种基于动态优先级地实时混合任务调度算法[J]. 四川大学学报:自然科学版,2006,43(3):544.

[责任编辑:伍少梅]