·体系结构与软件技术· 文章编号：1000-3428(2017)12-0038-07 文献标志码：A 中图分类号：TP316.2

多核平台下分区操作系统的安全关键任务调度方法

朱怡安[a]，黄林林[a]，李 联[b]，罗殊彦[a]

（西北工业大学 a. 计算机学院；b. 软件与微电子学院，西安 710072）

摘 要： 多核环境中并发、资源共享和任务迁移等特性，导致分区操作系统的安全关键任务调度存在较大的不确定性。为此，提出一种针对多核平台分区操作系统的安全关键任务调度方法。判断系统是否包含安全关键任务并将其分为关键分区和非关键分区。在系统层通过资源划分的方式确保关键分区的独立运行，根据分区利用率确定最小资源上限，在保证安全关键任务调度可靠性的同时，提高系统的资源利用率。在分区层设计基于动态优先级的双模容错任务调度算法和基于二分搜索的最小任务优先级搜索算法，以提升系统的容错能力。实验结果表明，该方法能够提高分区操作系统的资源利用率和任务调度可靠性。

关键词： 多核处理器；分区操作系统；容错调度；资源划分；安全关键任务；分区映射

中文引用格式： 朱怡安，黄林林，李 联，等. 多核平台下分区操作系统的安全关键任务调度方法[J]. 计算机工程，2017，43(12)：38-44.

英文引用格式： ZHU Yian，HUANG Linlin，LI Lian，et al. Safety-critical Task Scheduling Method for Partitioned Operating System in Multi-core Platform[J]. Computer Engineering，2017，43(12)：38-44.

Safety-critical Task Scheduling Method for Partitioned Operating System in Multi-core Platform

ZHU Yian[a]，HUANG Linlin[a]，LI Lian[b]，LUO Shuyan[a]

（a. School of Computer Science；b. School of Software and Microelectronics，Northwestern Polytechnical University，Xi'an 710072，China）

【Abstract】 Due to the characteristics of concurrent，resource sharing and task migration in multi-core environment，uncertaionty has been introduced into the safety-critical task scheduling for partitioned operating system. In view of this，this paper presents a safety-critical task scheduling method for partitioned operating system in multi-core platform. This method divids the system partition into critical partition and non-critical partition by judging whether it contains safety-critical tasks. On the system layer，resource pre-allocated method is used to ensure the independent operation of the critical partitions. Moreover，it determines the minimum resource upper limit according to the partition utilization，so as to improve the resource utilization rate of the system while ensuring the reliability of safety-critical task scheduling. On the partition layer，a dual-mode fault-tolerant task scheduling algorithm based on dynamic priority and a minimum task priority search algorithm based on binary-search are proposed，which can enhance the fault-tolerance abality of the system. Experimental results show that the proposed method can guarantee the partitioned operating system to have higher resource utilization and reliability of task scheduling.

【Key words】 multi-core processor；partitioned operating system；fault-tolerant scheduling；resource partitioning；safety-critical task；partitioned mapping

DOI： 10.3969/j.issn.1000-3428.2017.12.007

0 概述

面对嵌入式系统功能需求的复杂化和多样性，设计者将多个安全关键任务集成在同一平台，构成了混合关键系统[1]。嵌入式分区操作系统因其具有错误隔离的特性，能够有效提高系统的可靠性，现已被广泛应用于各种安全关键系统中[2]。分区操作系统的任务调度方法研究也已获得越来越多的关注。

基金项目： 航空科学基金“混合关键任务调度算法及无锁同步机制研究”(20150753010)；民用飞机专项科研技术研究类项目“高可靠操作系统内核关键技术研究”(XJZ-2015-D-76)；陕西省重点研发计划重大重点项目“时间/事件混合触发的实时操作系统技术与应用研究”(2016MSZD-G-8-1)。

作者简介： 朱怡安(1961—)，男，教授，主研方向为嵌入式软件、物联网、云计算；黄林林(通信作者)，硕士研究生；李 联，研究员；罗殊彦，博士研究生。

收稿日期： 2016-11-15 **修回日期：** 2017-01-07 **E-mail：** dy_llhuang@163.com

文献[3]针对 ARINC653 的两层调度模型进行改进,提出一种单层调度模型,为系统的每个任务赋予全局统一优先级,并在系统中维护一个全局优先级调度队列,加快了安全关键任务的响应速度,提高了系统的资源利用率。但由于系统共用调度队列造成分区之间较强的耦合性,降低了系统的可靠性。文献[4]针对 ARINC653 分区调度算法中存在的空闲时间问题,提出一种采用优先级位图算法和二级调度机制的空闲时间分区共享算法,提高了系统的资源利用率,但需要在系统运行时进行动态分区调度,引入了调度的不确定性,容易造成系统不稳定。

目前,多核处理器系统凭借体积小、低功耗、低成本和运算能力强的特点获得了越来越多的关注。据估计,2017 年工业控制领域的多核使用率将达到 50%[5]。2015 年 8 月 AEEC 机构颁发的 ARINC653P1-4 标准中增加了对于多核处理器的支持,可见在多核平台下实现分区操作系统已是大势所趋。但由于多核环境下的并发、资源共享和任务迁移等特性,功耗快速变化很有可能超出处理器的功耗预算,从而超过其散热能力导致过热[6],分区操作系统在多核平台下的调度执行出现了大量的不确定性问题[7],无法保证安全关键任务的可靠执行,因此如何在多核平台上实现分区系统的可靠调度已经变得至关重要。

针对多核系统平台上分区操作系统的调度问题,文献[8]提出了一种多核混合分区调度算法,将系统中的分区类型分为紧急响应分区和普通分区,所有分区内的任务根据其优先级进行抢占式调度。这种方法虽然提高了系统对于安全关键任务的响应速度,但削弱了分区之间的时间隔离,增加了不同关键级别任务之间的干扰,并且需要进行大量的分区切换,容易造成系统不稳定和资源的严重浪费,增加系统的不确定性。

针对上述问题,本文改进 ARINC653 分区操作系统的调度方法,在多核平台下提出一种新的安全关键任务调度方法。为保证安全关键任务的执行,首先在系统层隔离部分 CPU 资源用于关键分区的调度;然后给出根据分区利用率确定最小 CPU 资源上限的方法,在保证安全关键任务调度可靠性的同时最大化系统的资源利用率;最后在分区层设计基于动态优先级的双模容错调度算法和基于二分搜索的最小任务优先级搜索算法,从而提升系统的容错能力和任务调度可靠性。

1　分区系统调度模型

1.1　分区和任务模型

为保证分区系统中的安全关键任务调度,本文扩展分区操作系统的定义,将传统的系统分区分为2 种类型:关键分区和非关键分区。关键分区是指运行安全关键任务的分区,在系统调度中优先保证关键分区的独立执行;非关键分区是指只运行普通任务的分区。

为进行分区内的容错调度,本文在分区内部根据软件容错模型[9]的设计思想提出主、副版本任务集概念:主版本任务集为系统无错误发生时系统调度的任务集合;副版本任务集为主版本任务相同功能的不同实现,用于在主版本任务发生错误时替代主版本任务执行。

假设分区操作系统包含 n 个分区,分别用 P_1, P_2,…,P_n 表示,它们共同组成分区操作系统的分区集合 S,分区的定义为 $P_i=(\tau^i,\overline{\tau^i},I_i^{\mathrm{p}},E_i^{\mathrm{p}},T_i^{\mathrm{p}})$,各个分区特征定义如下:

1)分区内主版本任务集 $\tau^i=\{\tau_1^i,\tau_2^i,\cdots,\tau_n^i\}$,表示第 i 个分区内无错情况下执行的任务集合。

2)分区内副版本任务集 $\overline{\tau^i}=\{\overline{\tau_1^i},\overline{\tau_2^i},\cdots,\overline{\tau_n^i}\}$,表示第 i 个分区内出错情况下的替代执行任务集合。

3)分区标识属性 I_i^p,表示分区的类别,其取值范围为 $\{Hi,Co\}$,其中:Hi 为关键分区;Co 为非关键分区。

4)分区执行时间 E_i^{p},表示在每一个主时间轴周期中第 i 个分区需要执行的时间。

5)分区的周期 T_i^{p},表示第 i 个分区产生的周期。

任务 τ_j^i 定义为 $\tau_j^i=(V_j,L_j,C_j,Z_j,D_j)$,其任务特征定义如下:

1)任务类型标识属性 V_j:表示任务的类别,取值范围为 $\{0,1\}$,其中:0 表示主版本任务;1 表示为容错版本任务。

2)任务优先级属性 L_j:用于任务在调度过程中确定任务的调度顺序。

3)任务的最差执行时间 C_j:表示第 i 个分区内第 j 个任务在最差情况下所需要的处理器执行时间。

4)任务的周期 Z_j:表示任务执行的最小间隔时间。

5)任务的死限时间 D_j:表示任务的最晚完成时间。

1.2　调度模型

假设一个多核系统有 m 个处理器($m>1$),其集合为 $\{M_1,M_2,\cdots,M_m\}$,且这些处理器为具有相同处理能力的同构处理器集。本文多核分区系统调度方法模型如图 1 所示。在进行分区调度时,划定专用的处理器核资源用于关键分区的任务调度,剩余资源用于非关键分区的任务调度。本文的分区调度模型符合 ARINC653 标准规定的两层调度模型。在系统层,为每一个处理器核 $M_i(1\leqslant i\leqslant m)$ 单独维护一张分区调度表,其结构如图 2 所示,并采用轮转调度的方式进行单核分区的调度;在分区层,根据不同的分区类型采用不同的调度策略。在关键分区内采用本文提出的双模容错调度算法进行任务调度,保证

系统的可靠性;非关键分区采用单调速率(Rate Monotonic,RM)调度算法,增加系统的调度效率。每一个分区内部的任务只能在当前分区处于激活状态才可能被执行。

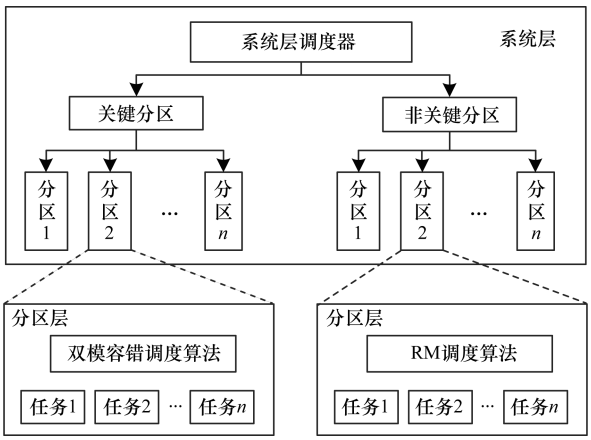

图 1 分区操作系统调度模型

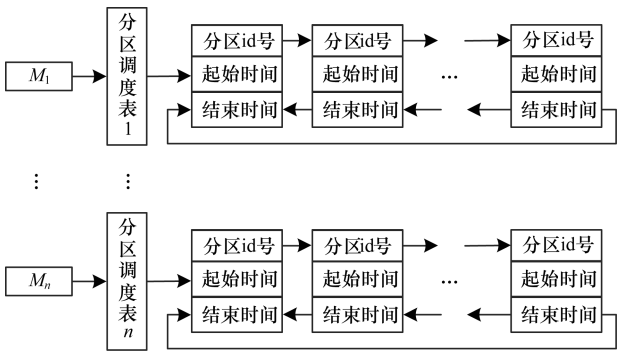

图 2 多核平台分区系统分区调度表结构

为保证在多核平台下调度的可靠性,系统行为需要遵守以下调度原则:

1)在分区调度层,分区内部的任务作为一个整体进行调度,不允许对分区内部的任务进行分拆,以保证分区的隔离性。

2)由于多核系统具有并行执行的特性,致使任务的最坏执行时间难以估算,且处理器核之间的迁移会导致其执行状态具有不可控性,因此不允许分区和分区内的任务进行核间迁移。

3)系统级的分区调度表是静态生成的,必须在系统启动之前完成分区调度表的创建,不可进行动态修改。

2 算法实现

本文调度方法的具体步骤如下:

步骤 1 处理器核划分。本文采用资源划分的方法进行分区的硬件隔离,根据关键分区利用率确定在分区可调度情况下的最小处理器资源需求,划分专用处理器资源用于关键分区的调度,剩余资源用于非关键分区的调度。

步骤 2 分区映射。将分配任务集到多个处理器核上的问题是一个类装箱问题,其已被证明是一个 NP 难度的问题。本文将对多种装箱策略进行研究,确定分区到处理器核之间的映射关系。

步骤 3 确定用于关键分区内部调度的双模容错调度方法。本文针对关键分区提出一种基于动态优先级的容错调度方法,并通过最小优先级搜索算法确定任务的动态优先级,保证系统在安全关键任务发生错误时,能够具有较强的系统自恢复能力,增强系统的可靠性。

2.1 处理器核划分

首先定义每个任务的利用率 $u_i = C_i/Z_i$,分区的利用率 u_{Pi} 为分区内任务 $\{\tau_1,\tau_2,\cdots,\tau_n\}$ 的利用率相加之和,即 $u_{Pi} = \sum_{i=1}^{n} u_i$,则分区操作系统的关键分区的整体利用率为 $u_{Hi} = \sum_{L\in Hi} u_{Pi}$。

在任务调度过程中,任务集可调度的必要条件为系统的计算能力大于任务集所需要的处理能力[10]。假设单处理器的最大计算能力为 1,则有 m 个同构处理器的多核系统的最大计算能力为 m。为了保证任务的可调度性,任务集需要的处理能力必须小于 m。

因此,为了保证关键分区的可调度性,需要保证处理器可用资源大于关键分区调度所需处理能力,即需要满足公式 $m_1 \geqslant u_{Hi} \times f$,式中:$m_1$ 为调度任务集所需的最小处理器数量;f 为可调度因子,f 的计算公式为 $f = u_O/u_A$,其取值范围为 $[1,\infty)$,u_A 为算法 A 可调度的任务集的最大处理器利用率,u_O 为最佳调度算法的可调度的任务集的最大处理器利用率。由于本文进行分区层调度的方法均采用基于 RM 的调度策略,而 RM 调度算法已被证明为最优静态调度算法,因此有 $u_A \approx u_o$,由此可得 f 近似取值为 1。

定义利用率边界函数 $h(t)$,其值为在时间区间 $[0,t)$ 内需要执行的最大任务数量,其中 t 的取值从 0 到任意最大值,其计算公式如式(1)所示。

$$h(t) = \sum_{i=1}^{n}\left\{\max\left(0,\left[\frac{t-D_i}{Z_i}\right]+1\right)\right\}C_i \tag{1}$$

则处理器的负载最大值为在任意时间区间内需要的最大处理器计算能力,计算公式如式(2)所示。

$$load(\tau) = \max_{\forall t}\left(\frac{h(t)}{\Delta t}\right) \tag{2}$$

为确保任务调度的顺利进行,在任务调度过程中必须保证每个时间节点都有足够的处理器资源可以使用,否则就会发生任务超时异常。因此,为关键分区分配的处理器核数量需要满足如式(3)所示的必要条件。

$$load(\tau) \leqslant m_2 \tag{3}$$

因为分区被作为一个整体进行调度,所以需要消除因大利用率分区导致的调度问题。例如 3 个关键分区的利用率值均大于 0.5,则无法在 2 个核上完成调度,因此,处理器核的分配需满足式(4)所示的条件。

$$n_{Hi}(u_{pi}>0.5)\leqslant m_3 \tag{4}$$

最后根据 $m=\max(m_1,m_2,m_3)$ 计算需要的处理器核数,从而确定进行关键分区任务调度所需要的最少处理器核数 $N_{Hi}=m$,则用于非关键分区调度的处理器核数量 $N_{Co}=n-m$,其中 n 为所有处理器数量。

2.2　分区映射

分区映射是指为分区分配用于调度的处理器资源。为了确保分区映射的顺利进行,规定每一个分区仅与一个特定的处理器核对应,并且在执行过程中不能进行核间的分区迁移。关键分区映射到处理器集合 N_{Hi} 中,而非关键分区映射到处理器集 N_{Co} 中。通过这种方法把分区映射问题简化为 $n:m$ 的整体装箱问题,利用装箱策略进行处理器核的分配。

装箱问题已被证明是一个 NP 问题[11]。针对装箱问题的解决方法只能采用启发式算法进行近似求解。目前存在的解决算法主要有首次适配(First-Fit)算法、最佳适配(Best-Fit)算法和最差适配(Worst-Fit)算法。

为了保证所选择装箱算法的准确性,本文通过实验验证使用不同装箱策略的 CPU 利用率和可调度性之间的关系。在实验中,设置处理器核数为 4,分区数量为[4,20],每个分区包含[3,15]个任务,每个任务的周期从{10,20,40,50,100,200,400,600,800,1 000}中随机选择(周期单位为 ms),任务的死限时间与周期相同,任务的执行时间为任务死限时间的 30% ~ 80% ,其具体取值根据随机生成算法进行选择,共进行10 000 次实验。实验结果如图 3 所示。

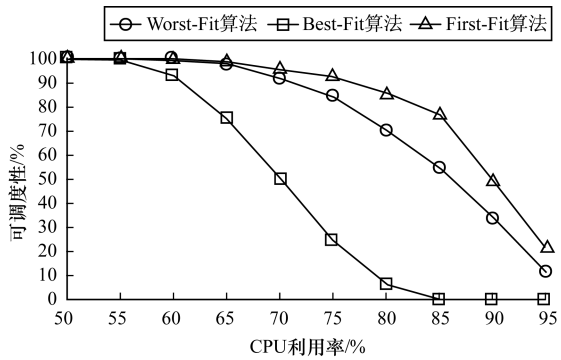

图 3　不同装箱算法 CPU 利用率与可调度性的关系

为防止处理器空转,充分利用多核平台计算能力强的优势,需要尽可能保证多核平台的负载均衡。针对 3 种不同的装箱算法,本文验证了任务集数量和负载差异(最大 CPU 负载和最小 CPU 负载之差)之间的关系,实验结果如图 4 所示。

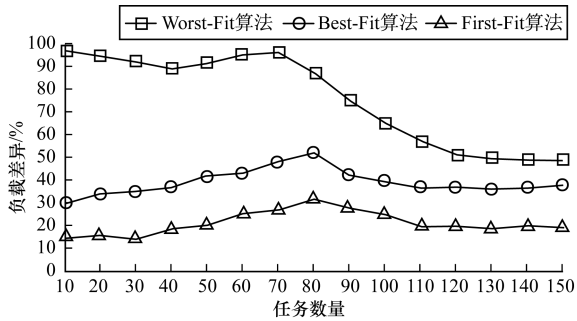

图 4　任务集数量与负载差异的关系

由实验结果可知,最差适配算法在保证可调度性的同时能够最大化系统资源的利用率。在逐渐增大资源利用率的过程当中,首次适配算法获得了更高的可调度性。因此,针对分区调度的侧重点不同,为了保证关键分区的可调度性,同时保证系统整体的资源利用率,针对关键分区的映射采用最差适配算法,而非关键分区则采用首次适配算法,能够从总体上提升系统的资源利用率。

在完成本步骤之后,会形成处理器核与分区之间的 $n:1(n\geqslant 1)$ 绑定关系,保证每一个分区与一个特定的处理器绑定,根据处理器核心与分区的映射关系形成自然的调度域,其关系如图 5 所示。

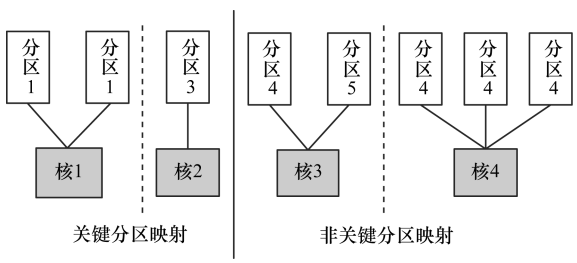

图 5　处理器与分区映射的关系

2.3　双模容错算法

双模容错调度算法的基本设计思想是在系统中优先调度主版本任务,副版本只有在主版本任务发生故障的情况才会启动运行。在容错模型中,只要保证2 个版本中的一个任务在时限之内完成即可[12],从而增加系统的可靠性。

为了保证分区系统可以使用双模容错调度算法进行分区内任务的调度,在分区内部维护 2 个任务队列:调度队列和容错队列,2 个队列均按照优先级从高到低进行排列。在进行任务创建时,需要同时

创建主版本任务和副版本任务控制块,主、副版本任务控制块除了 V 参数不同之外,具有相同的优先级、执行周期和最差执行时间。在系统无错误发生时系统依据固定优先级抢占策略从调度队列中选择最高优先级任务。在系统发生错误时,停止出错主版本任务的运行,并从调度队列中移除主版本任务的任务控制块,将其 V 参数改为 1,并在副版本任务队列中找到相应的任务控制块,修改其 V 参数为 0,标识其为主版本,把其加入调度队列中,继续进行任务的调度。

在任务发生错误时,一般已经耗费了一定的运行时间,如果单纯采用继承主版本优先级继续进行任务调度,其副版本重新运行的任务响应时间必定大于无错误发生情况下的任务响应时间,可能会导致任务无法在死限时间到来之前完成任务的执行[13]。如果安全关键任务发生此种错误,将造成严重的系统错误,致使整个系统无法运行。针对这一问题,本文的容错调度方法在安全关键任务发生错误时首先进行容错任务的响应时间计算,若其响应时间大于任务的死限时间,则允许容错任务进行优先级提升,保证任务响应时间小于其死限时间。若其响应时间小于任务死限时间,则直接继承主任务的优先级,将出错任务对系统的影响降至最小。

假设 P_i 分区内发生安全关键任务执行错误,为了保证出错任务的时限要求,首先需要重新计算出错任务的响应时间 $R_i = S_i + C_i + PB_i$,S_i 为任务 $\overline{\tau_i}$ 的最晚开始时间,其计算公式为 $S_i = \sum_{L_j > L_i} C_j$,$PB_i$ 为在任务执行完成之前在分区调度表中不属于本分区的运行时间,$PB_i = \sum_{s<j<n \& L_i \neq L_j} e_j - s_j$,$s$ 为出错任务开始运行时的分区调度表项,n 的计算公式为 $\sum_{s \leqslant j \leqslant n} e_j - s_j > \sum_{L_j \geqslant L_i} C_j$,取最小值。若 $R_i < D_i$,则对出错任务的副版本任务 $\overline{\tau_i}$ 赋予与主版本任务相同的优先级;若 $R_i > D_i$,则需要进行优先级的提升操作。由于在分区系统内部,调度队列按照优先级递减的方式有序排列,为减少由于优先级提升所造成的时间开销,本文在进行优先级的搜索时,充分利用调度队列的有序特点,采用二分查找的方式搜索最优优先级。首先假设提升后的任务优先级为 $\overline{L_i} = L_i$,其搜索算法包含以下步骤:

步骤 1 假设分区内任务最高优先级为 $L_{\max}$,最小搜索优先级为出错任务优先级 $L_{\min} = L_i$,则依据二分搜索原则,设提升后的优先级为:$L_{\text{tmp}} = \lceil (L_{\min} + L_{\max})/2 \rceil$。

步骤 2 计算分区内所有优先级低于 L_{tmp} 的任务的响应时间,保证其能够满足任务的死限要求。假设发生错误的任务为 τ_i,错误发生时任务 τ_i 已执行的时间为 t_i,为了得到任务的已执行时间,本文方法改进了任务控制块,在任务控制块中添加已执行时间的参数,对所有优先级小于 L_i 的任务 τ_j 其最坏响应时间为 $R_j = S_j + C_j + PB_i + t_i$。对于所有优先级在 (L_i, L_{tmp}) 中的任务 τ_j,其任务响应时间分 2 种情况:

1)若 n 的计算数值不变,则新的任务响应时间 $R_j' = R_j + C_i$。

2)若 n 发生变化,则新的任务响应时间计算公式为 $R_j' = R_j + C_i + \sum_{n<j<e} e_j - s_j$,其中 e 的取值为公式 $\sum_{n \leqslant j < e \& T_j^p = pid} e_j - s_j \geqslant C_i$ 成立的最小值,pid 为当前运行的分区号,T_j^p 为分区调度表项的分区号。

步骤 3 若所有的分区内受影响任务的响应时间均低于其死限时间,则记提升后的任务优先级为 $\overline{L_i} = L_{\text{tmp}}$,$L_{\max} = L_{\text{tmp}}$,否则记 $L_{\min} = L_{\text{tmp}}$,循环进行上述步骤,直至分区内存在任务无法保证其死限时间或 $L_{\text{tmp}} = L_i$。

循环结束后若 $\overline{L_i} \neq L_i$,则完成优先级提升操作,且副版本任务的优先级提升为 $\overline{L_i}$,否则给出无法完成容错调度的结论。

3 实验验证

为验证本文调度方法的正确性,针对多方面进行实验验证。

本文所有的验证实验均基于以下假设:处理器核数为 8,系统的整体 CPU 资源利用率区间为 [1.0,7.0],关键分区数量为 n_1,非关键分区数量为 n_2,每个关键分区的总体资源利用率为 [0.2,0.6],非关键分区的总体利用率为 [0.1,0.4];单个任务的周期取值为 {10,20,40,50,100,200,400,600,800,1 000}(单位为 ms),任务的死限时间处于任务周期的 10% ~90% 之间,任务的最坏执行时间为任务死限时间的 20% ~80% 之间。

实验数据选择方案:首先在设定范围内随机选择关键分区数量为 n_1,依次利用随机数生成函数为 n_1 个分区在 [0.2,0.6] 生成相应的分区利用率 u_{pi},并利用本文方法的处理器核划分算法计算相应的处理器

核数量 n_k,利用步骤 2 的分区映射方法进行分区映射;然后循环利用随机生成算法选择单任务的利用率,直至所有任务的利用率之和 $u_{pi}-u_{sum}<0.05$。

3.1　处理器核划分实验

进行本实验的目的是验证处理器核分配的合理性,保证分配给关键分区的处理器核为可调度情况下的最小值,只有在这种情况下才能最大化资源利用率,保证用于非关键分区调度的处理器核的最大值和系统的资源利用率。

实验方案如下:选取关键分区数量为[2,20],共进行 1 000 次实验,得到处理器核数 N_{Hi} 为 n_k 和 n_k-1,n_k+1 的可调度性对比,如图 6 和图 7 所示。可以看出,随着关键分区数量的增加,由本文处理器核划分方法分配的处理器核数量能够满足系统关键分区调度需求。在此基础上增加处理器核数量无法明显提高系统的可调度性,而减少处理器核数量将明显降低系统的可调度性。

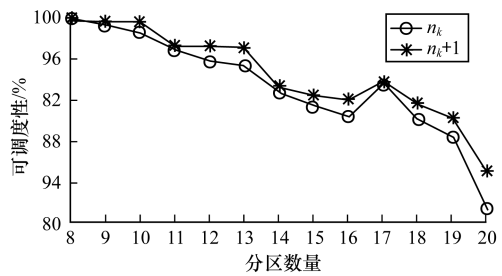

图 6　处理器核数为 n_k 和 n_k+1 时的可调度性对比

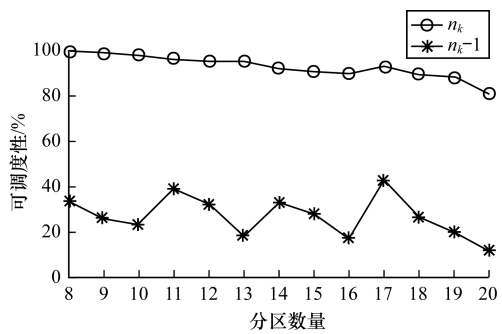

图 7　处理器核数为 n_k 和 n_k-1 时的可调度性对比

3.2　多核分区调度实验

进行本实验的目的是验证本文调度方法对于系统利用率的影响,对比实验算法为一种混合关键调度算法[14],该算法对关键任务和非关键任务进行统一的处理器核分配,在发生故障时,挂起全部的非关键任务保证关键任务的可靠运行。

实验方案如下:

1)2 个算法均在无错的情况下运行,计算系统的整体资源利用率。假设系统中分区数量为[5,35],通过实验数据选择方案生成实验数据,并保证实验数据中高关键级分区数量占整个分区数量的 10% ~50%,最少为 1 个关键分区,并选取 1 000 组实验数据进行实验,对实验数据进行平均,其系统资源利用率与分区数量关系如图 8 所示。可以看出,在无错情况下,随着分区数量增加,本文方法相对混合分区调度方法能够保持相当的资源利用率。

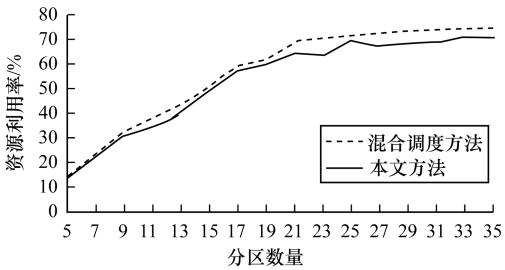

图 8　分区数量与资源利用率的关系

2)假设分区数量为 30,2 种调度方法均在系统运行 1 000 ms 注入一次低关键级分区故障,且故障发生的处理器核随机选择,其他数据与实验方案 1)相同,混合分区调度算法在发生非关键级分区错误后将放弃处理器核上所有的非关键级分区,其系统资源利用率与关键分区数量对比关系如图 9 所示,其中,N 代表无错误出现情况,Y 代表有错误出现的情况。可以看出,在发生非关键分区错误的情况下,本文提出的调度方法能够保证系统仍处于较高的利用率,而混合分区调度方法则会发生较大的利用率下降。实验结果表明,本文方法能够保证系统在各种情况下都处于较高的利用率。

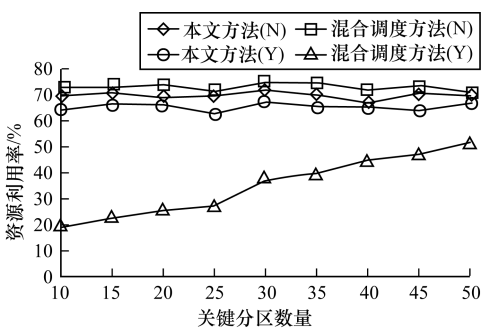

图 9　资源利用率与关键分区数量的关系

3.3　分区容错调度实验

进行本实验的目的是验证本文分区内双模容错算法的容错能力。电子计算机典型的“平均无故障间隔时间”约为10^4 h,本实验假设关键分区

数量在[2,20]之间,任务的最坏执行时间为任务死限时间的 40% ~80% ,关键分区内部故障发生时间间隔为 2 000 ms,对比容错调度算法为优先级不变的简单容错调度算法[15],该算法的容错思想为错误发生之后重新开始运行任务,不改变任务的优先级。

实验方案如下:分区的生成方法依据实验数据选择方案,使用双模容错调度算法和简单容错算法分别进行调度,共进行 1 000 次实验,其平均容错成功率和错误发生的时间点(任务发生错误的时间与最坏执行之间比例)之间的关系如图 10 所示。可以看出,本文采用的双模容错算法,能够通过优先级提升的方式保证较高的关键容错能力。相较于简单容错算法,在任务执行后期发生错误的情况下,本文方法能够提高 50% 左右的容错能力,大幅提升了系统任务调度的可靠性。

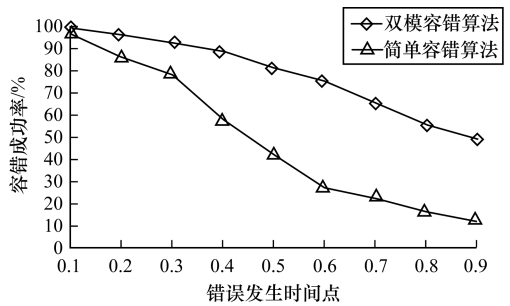

图 10 容错成功率与错误发生时间点的关系

4 结束语

本文以提高多核环境下分区操作系统安全关键任务调度的可靠性和系统资源利用率为目的,提出一种新的分区操作系统关键任务调度方法。该方法将系统分为 2 个集合:关键分区和非关键分区。关键分区重点保障安全关键任务执行的可靠性,非关键分区重点提升系统资源的利用率。实验结果表明,本文方法能够在保证系统资源利用率的同时,使分区操作系统具有更高的可靠性。下一步将对分区层的动态调度方法进行研究,以更好地完善整个分区操作系统的关键任务调度方法。

参考文献

[1] BURNS A,DAVIS R. Mixed Criticality Systems——A Review[D]. York, UK: Department of Computer Science,University of York,2013.

[2] HORVATH G,CHUNG S,DVORAK D,et al. Safety-critical Partitioned Software Architecture: A Partitioned Software Architecture for Robotic Spacecraft[C]// Proceedings of Infotech@ Aerospace Conference 2011. [S.l.]:AIAA,2011:1-12.

[3] TRUJILLO S, CRESPO A, ALONSO A, et al. MultiPARTES:Multi-core Partitioning and Virtualization for Easing the Certification of Mixed-criticality Systems[J]. Microprocessors and Microsystems,2014, 38(8):921-932.

[4] NELSON A,NEJAD A B,MOLNOS A,et al. CoMik:A Predictable and Cycle-accurately Composable Real-time Microkernel[C]//Proceedings of Design, Automation and Test in Europe Conference and Exhibition. Dresden, Germany:[s. n.],2014:222.

[5] GU C, GUAN N, DENG Q, et al. Partitioned Mixed-criticality Scheduling on Multiprocessor Platforms[C]// Proceedings of the 20th IEEE Conference on Real-time and Embedded Technology and Applications Symposium. Washington D. C. ,USA:IEEE Press,2014:292.

[6] 邱晓志,安 虹,陈俊仕,等. 一种功耗受限情况下的多核处理器能效优化方案[J]. 计算机工程,2017, 43(4):39-45.

[7] BURNS A, BARUAH S. Towards a More Practical Model for Mixed Criticality Systems[C]//Proceedings of the 34th IEEE Real-time Systems Symposium. Vancouver,Canada:[s. n.],2013:1-6.

[8] 郝继锋,虞保忠,周 霆,等. 一种多核混合分区调度算法设计与实现[J]. 微电子学与计算机,2016, 33(7):140-144.

[9] SREEKUMAR A, SWETHA K, SWETHA A, et al. Enhanced Performance Capability in a Dual Redundant Avionics Platform-Fault Tolerant Scheduling with Comparative Evaluation[J]. Procedia Computer Science, 2015,46:921-932.

[10] AL-BAYATI Z,ZHAO Q,YOUSSEF A,et al. Enhanced Partitioned Scheduling of Mixed-criticality Systems on Multicore Platforms[C]//Proceedings of the 20th Asia and South Pacific Design Automation Conference. Washington D. C. ,USA:IEEE Press,2015:630-635.

[11] DAVIS R I, BURNS A. Hierarchical Fixed Priority Pre-emptive Scheduling[C]//Proceedings of the 26th International Real-time Systems Symposium. Washington D. C., USA:IEEE Press,2005:398.

[12] 丁万夫,郭锐锋,秦承刚,等. 容错优先级可提升的抢占阈值容错调度算法[J]. 软件学报,2011,22(12): 2894-2904.

[13] BARUAH S K, BURNS A, DAVIS R I. Response-time Analysis for Mixed Criticality Systems[C]//Proceedings of the 32nd Real-time Systems Symposium. Washington D. C., USA:IEEE Press,2011:34-43.

[14] BARUAH S, CHATTOPADHYAY B, LI H, et al. Mixed-criticality Scheduling on Multiprocessors[J]. Real-Time Systems,2014,50(1):142-177.

[15] KOWALSKI D R, WONG P W H, ZAVOU E. Fault Tolerant Scheduling of Non-uniform Tasks Under Resource Augmentation[C]//Proceedings of the 12th Workshop on Models and Algorithms for Planning and Scheduling Problems. La Roche-en-Ardenne, Belgium: [s. n.]:2015:1-3.

编辑 金胡考