软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(6):1681−1698 [doi: 10.13328/j.cnki.jos.005470]

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

考虑中断和上下文切换开销的响应时间分析*

于广良[1], 杨孟飞[2]

[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)

通讯作者: 于广良, E-mail: ygl_222@126.com

摘 要: 实时嵌入式系统多采用中断和上下文切换实现多任务间调度,在对此类系统进行可调度性分析时,在任务的最差响应时间计算中必须包含中断和上下文切换开销.现有包含这些开销的方法是将中断作为高优先级任务,同时将上下文切换开销加入到任务最差执行时间中进行分析,然而这些方法过于粗略,缺乏对实际系统细节的考虑,计算得到的最差响应时间并不精确.首先,对中断和上下文切换的机制和时间流程进行详细的阐述,进而分析中断和上下文切换对任务关键性时刻的影响;接着,给出包含上述开销的更加精确的响应时间计算方法;最后进行仿真验证.扩展了包含系统调度开销的响应时间计算方法,可用于资源受限的硬实时系统中需要精确计算响应时间的场合.

关键词: 实时系统;嵌入式系统;可调度性;最差响应时间;中断;上下文切换

中图法分类号: TP311

中文引用格式: 于广良,杨孟飞.考虑中断和上下文切换开销的响应时间分析.软件学报,2018,29(6):1681−1698. http://www.jos.org.cn/1000-9825/5470.htm

英文引用格式: Yu GL, Yang MF. Response time analysis of embedded systems by taking interrupt and context switch overheads into account. Ruan Jian Xue Bao/Journal of Software, 2018,29(6):1681−1698 (in Chinese). http://www.jos.org.cn/1000-9825/5470.htm

Response Time Analysis of Embedded Systems by Taking Interrupt and Context Switch Overheads into Account

YU Guang-Liang[1], YANG Meng-Fei[2]

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

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

Abstract: Interrupt and context switch are basic mechanisms for multi-task scheduling in real-time and embedded systems. During schedulability analysis, the overheads of the interrupt and context switch should be considered in the calculation of tasks' worst-case response time. The current calculation methods of response time add interrupt as task with high priority, and simply add the overheads of context switch in the meantime. However, these methods neglect the details of practical systems and roughly give an inaccurate worst-case response time. In this paper, the mechanisms and time flow of interrupt and context switch are thoroughly described. In addition, their influence on tasks' critical instant is discussed. More importantly, a much more accurate calculation method for response time is presented. At last, simulations is conducted to validate the improvement in accuracy of this new method. The response time analysis accounting for scheduling overheads is extended in this paper for the resource-constrained hard real-time systems which need to accurately calculate the response time.

Key words: real-time systems; embedded systems; schedulability; worst-case response time; interrupt; context switch

* 基金项目: 国家自然科学基金(91118007, 61502031, 61632005)
Foundation item: National Natural Science Foundation of China (91118007, 61502031, 61632005)
本文由形式化方法的理论基础专题特约编辑傅育熙教授、李国强副教授、田聪教授推荐.
收稿时间: 2017-07-01; 修改时间: 2017-09-01; 采用时间: 2017-11-06; jos 在线出版时间: 2017-12-28
CNKI 网络优先出版: 2017-12-29 13:19:31, http://kns.cnki.net/kcms/detail/11.2560.TP.20171229.1318.011.html

硬实时系统(hard real-time systems)是对时间要求最苛刻的实时系统,任务一次错过截止时间便会导致整个系统的错误,甚至发生灾难性的事情[1].这种系统一般使用嵌入式硬件系统实现,其上运行嵌入式实时操作系统(real-time operating system,简称 RTOS)对任务进行调度.在系统设计时,为保证可信性[2],通常要对系统中的任务进行可调度性分析,以确保所有任务都能够在截止时间之前完成.判断系统的可调度性通常使用可调度性测试(schedulability test).

响应时间分析(response time analysis,简称 RTA)是最常用的可调度性测试之一,它通过计算任务最差响应时间(worst-case response time,简称 WCRT),与任务截止时间相比较,判断系统任务的可调度性.响应时间分析由 Joseph 和 Pandya[3]在 1986 年及 Audsley 等人[4]在 1993 年给出,得到了学术界广泛的研究和工业界的应用.利用这种方法对实际嵌入式系统进行分析时,还应考虑系统的中断和任务间调度引起的上下文切换开销.虽然这些开销与任务最差执行时间(worst-case execution time,简称 WCET)相比要小得多,然而对其研究仍有理论和现实意义,如:

1) 在任务最差响应时间的计算公式中,抢占次数的计算采用向上取整运算,这意味着低优先级任务的执行被很小时间的延迟就足以导致高优先级任务更多的抢占,从而引起 WCRT 的大幅增长;
2) 在其他方面,如 WCET 的估计精度难以进一步提升时,对这些开销的估计显得同样重要;
3) 对系统细节的分析,可以增强响应时间分析在实际系统中应用的信心.

目前已有一些尝试在响应时间分析中包含上述开销的研究,综述如下.

- 中断的研究

Regehr[5]详细讨论了实时嵌入式软件里的中断,并指出需要验证系统内中断过载问题.Brylow 和 Palsberg[6]分析了使用汇编语言编写的中断驱动程序,提出了检查每个中断是否都能在截止时间前处理完成的方法.Jonathan 等人[7]利用上下文切换次数界限和基本路径来减小需搜索的路径空间,将中断驱动程序转化为顺序程序进行分析,以得出主程序在多个中断服务程序影响下的最差执行时间.他们的转化过程需考虑较多路径,故分析的复杂性较高.Leyva-del-Foyo 等人[8]提出了将任务与中断管理进行集成,定义统一的优先级空间进行调度.Jeffay 等人[9]设计了解决动态优先级系统里包含中断处理开销的可调度性分析问题的方法,并且简单讨论了静态优先级系统的情况.Sandström 等人[10]讨论了用于汽车系统运动控制的实时系统开发时遇到的静态调度与中断集成的问题,提出可以将中断当作高优先级的任务对待.Brandenburg 等人[11]综述了多处理器实时系统的中断开销计算方法,包含了对单处理器系统的讨论.他们总结了 3 种包含中断服务程序的方法:以时间片为中心的包含方法、以任务为中心的包含方法和以处理器为中心的包含方法.另外,有相关的工作研究了具体的 RTOS 中的开销,如文献[12,13].在上述研究中,中断服务程序多基于简单模型,对实际系统中中断与任务的关系缺乏讨论.

- 上下文切换开销的研究

Katcher 等人[14]扩展了固定优先级调度下的内核开销计算方法.随后,Burns 等人[15]针对时钟调度(tick scheduling),分析了调度器的开销和任务从延时队列移动到运行队列花费的时间开销,建立了实时系统调度器更为精确的模型.Gabriëls 等人[16]综述了包含时钟中断和上下文切换开销的相关技术.另一方面,Echagüe 等人[17]提出了用于离线计算周期性任务在采用最优静态和动态优先级分配策略的调度器下准确抢占次数的模型.Yomsi 等人[18]通过在分析中考虑每个抢占自身的开销,改进了 Echagüe 等人的模型.他们指出:如果考虑抢占开销,关键性时刻(critical instant)[19]并不出现在所有任务一齐释放时.然而,他们在其后计算中仍采用了任务一齐释放的假设,在本文第 3.3 节将证明这是不安全的.上述研究中,文献[14–16]主要讨论的是抢占造成的开销、基于上下文切换与时钟中断的简单模型,文献[17,18]讨论的是抢占次数的计算问题.在计算中,都采用任务同时释放的假设,没有考虑任务间的释放偏移.

综上所述,对于响应时间分析中包含中断和上下文切换等时间开销的研究,存在两方面不足.一是对中断与任务的关联(如任务由中断释放)情况缺乏讨论;二是在计算中多采用任务同时释放的假设,没有考虑任务间释放偏移.针对这些不足,本文的主要工作如下:提出任务由中断释放时,如何在其响应时间计算里包含系统中的中断开销;指出了包含上下文切换开销时,任务间释放偏移对任务的关键性时刻的影响;给出了一种比现有方法

更加精确的包含上下文切换开销的响应时间计算方法;仿真了真实的软硬件环境下包含中断开销的任务最差响应时间,并对遇到的实际问题提出解决办法.

本文第 1 节对实时嵌入式系统中的中断和上下文切换机制进行详细的讨论.第 2 节给出系统软件模型并对已有的响应时间计算方法进行简单回顾.第 3 节详细讨论中断和上下文切换开销对于关键性时刻的影响.第 4 节提出一种包含上下文切换开销的任务最差响应时间计算方法.第 5 节进行仿真分析.最后,第 6 节总结全文,并指出后续研究方向.

1 概 述

1.1 中 断

一般来说,处理器的异常包含了中断、陷阱、故障和终止.陷阱、故障和终止通常是同步发生的,是执行当前指令的结果;中断通常是异步发生的,是来自处理器外部的 I/O 设备的信号的结果[20].一个典型的中断处理时间流程如图 1 所示.

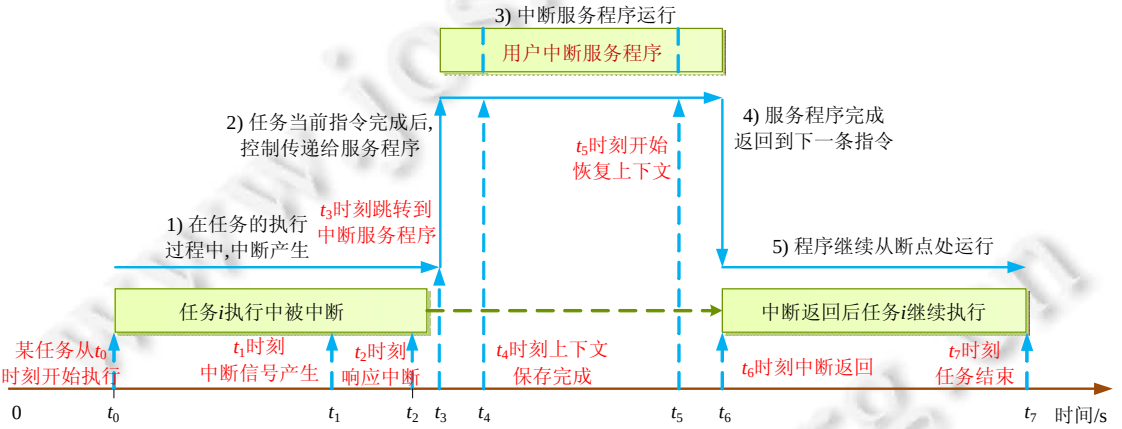

Fig.1 Timeline of interrupt handling

图 1 中断处理时间流程

任务程序在 t_0 时刻开始执行,t_1 时刻中断信号产生,t_2 时刻处理器开始响应中断,此时,处理器将继续完成正在执行的指令(有些处理器可能允许中断长指令),完成断点保存并跳转到中断向量表处,以上部分由处理器自动完成.从 t_3 时刻起,执行中断向量表中指令并分支到相应的中断服务程序,开始任务上下文的保存等,从 t_4 时刻到 t_5 时刻为用户编写的中断服务程序的执行时间,到 t_6 时刻恢复了被中断任务的上下文,中断服务程序处理完成,任务程序继续执行到 t_7 时刻结束.

为了使系统快速响应紧急事件,经常允许中断嵌套,以减小高优先级中断的响应时间,中断嵌套的典型处理流程如图 2 所示.

处理器刚进入中断时,往往会自动关闭中断以保证对断点的处理不被打扰.在进入用户中断服务程序前,可以打开高优先级中断以加快其响应.此时若有高优先级中断请求,系统将再次跳转到高优先级中断执行,执行完毕后返回到低优先级中断,低优先级中断继续执行,在返回任务前关中断,返回后处理器将自动打开中断(不同处理器处理流程或有差异).

中断通常是处理一些关键和紧急的情况,由于优先级高于任务,其处理时间应当越短越好,尽可能把更多的处理向后推迟,以减少对高优先级任务的干扰.某些时候,在用户中断服务程序中仅仅发布信号或者缓存数据(上半部),然后将更多的处理操作放到任务中完成(下半部).这样,中断可能使得某个比被中断的任务优先级更高的任务进入就绪态,则在中断服务程序结束后将进行任务切换,运行更高优先级的任务,使用符号 $\gamma_{i,j}$ 来表示这

种情况下中断下半部处理的时间开销,如图 3 所示.这种情况下的中断处理时间流程如图 4 所示.

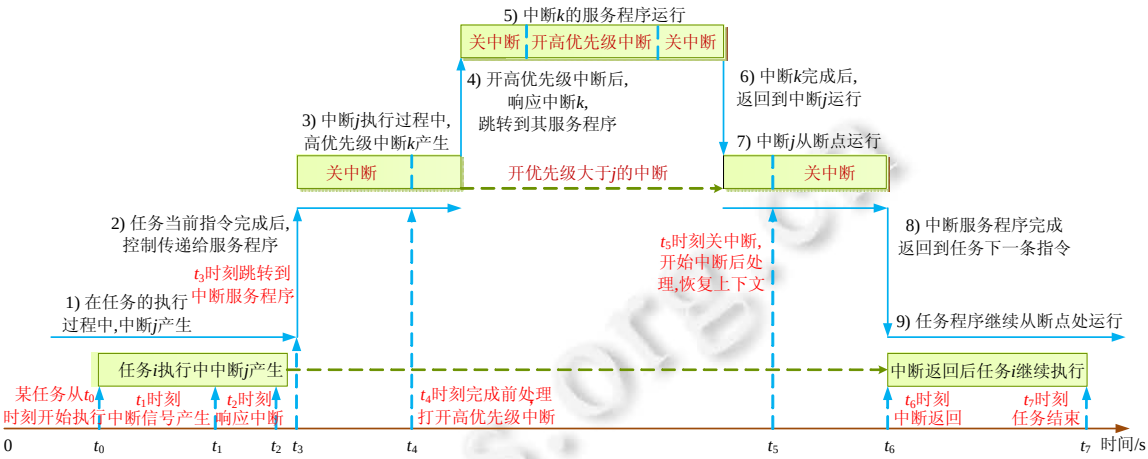

Fig.2 Timeline of interrupt nested

图 2 中断嵌套时间流程

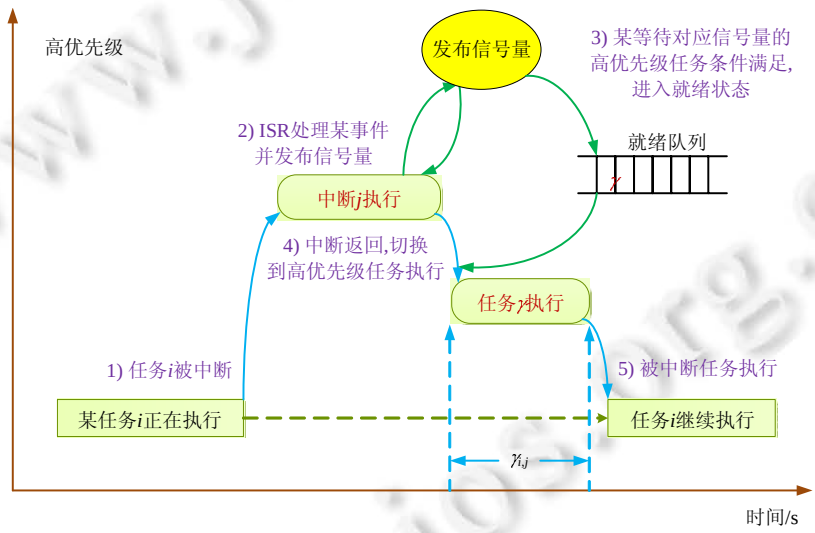

Fig.3 Task released by interrupt and causes preemption

图 3 中断上下部模型

图 4 与图 1 的时间流程有所不同,在 t_5~t_6 时间段里,服务程序检测到有比当前任务更高优先级的任务就绪,将进行任务切换,中断返回后高优先级任务得到执行,当其执行结束后,由系统调用切换到内核恢复先前被中断的任务继续执行.

事实上,这也是中断释放任务的典型处理过程.系统中的周期性任务通常通过系统的时钟中断来释放,例如:图 4 的中断若为系统的时钟中断,则其在某设定的时间点被触发(即任务到达),到达的任务在中断服务程序中被释放,中断返回后开始执行.其他形式的中断释放任务的情况与此类似.这种情况下,系统中的其他中断与释放某任务的中断可能形成嵌套,造成某任务释放延迟,在任务最差响应时间分析中需要考虑.

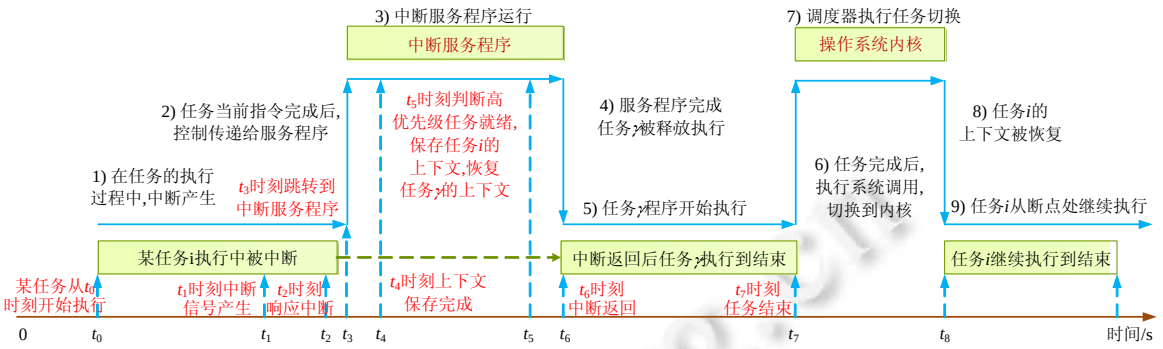

Fig.4 Timeline of interrupt handler and context switches

图 4 中断上下部时间流程

1.2 上下文切换

操作系统内核使用一种称为上下文切换(context switch)的较高层形式的控制流来实现多任务,内核为每个任务维持一个上下文.上下文就是内核重新启动一个被抢占的任务所需的状态,它由一些对象的值组成,这些对象包括通用目的寄存器、浮点寄存器、程序计数器、用户栈、状态寄存器、内核栈和各种内核数据结构等[20].上下文切换就是 CPU 从一个任务切换到另一个任务,它的产生有很多原因,可能是任务被抢占或任务用尽了自己的时间片等.上下文切换包括了保存旧的状态和恢复新的状态.通常情况下,CPU 拥有的寄存器越多,切换带来的开销就越大.由于抢占导致的上下文切换的实例如图 5 所示.

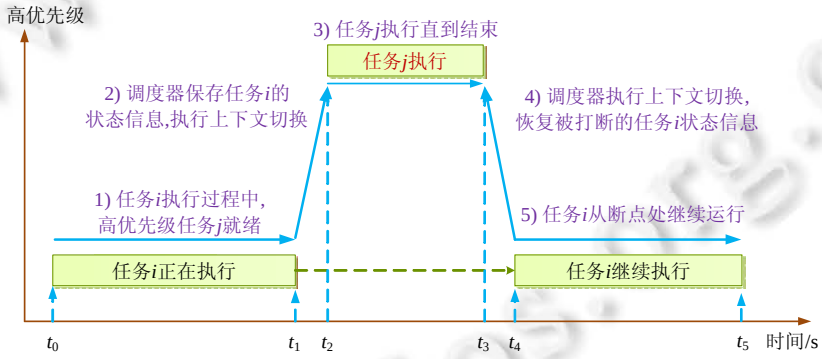

Fig.5 Example of context switch between tasks caused by preemption

图 5 抢占导致的上下文切换示意图

用τ表示任务,在任务τ_i执行过程中,高优先级的任务τ_j被释放,则系统调度器将执行上下文切换,首先保存任务τ_i的状态信息,以便后面恢复;然后加载任务τ_j的状态信息,切换到τ_j运行,等其运行完毕后,调度器将重新恢复任务τ_i的状态信息,令其从断点处继续运行,好像没有被打断一样.

任务的到达和释放一般是通过某些事件来触发的,对于周期性任务而言,可能是系统定时器的定时结束,也可能是某个外部周期性信号的触发,如果任务τ_j是通过中断释放的,那么它的时间处理流程可能如图 4 所示.图 5 中的时间区间 t_1~t_2 相当于图 4 中的 t_2~t_6,其中包含了中断的处理时间和上下文切换的时间;图 5 中的时间区间 t_3~t_4 相当于图 4 中的 t_7~t_8,其中包含了上下文切换的时间.

图 5 中的 t_1~t_2 和 t_3~t_4 分别代表了从τ_i切换到τ_j和从τ_j切换回τ_i所用的时间.通常情况下,这两部分时间是不相同的,若τ_j是新释放任务,则前一部分包含了τ_j的释放时间(任务创建或任务加入就绪队列的时间).进一步,如果τ_j由中断释放,那么由图 4 看出,前部分时间还同时包含了中断的处理执行时间.为了简化分析,本文用t_{sw}代表

切换开销的最大值,即图 5 中的 t_1~t_2 和 t_3~t_4 两者中切换代码 WCET 的较大者.如果是中断释放的任务,如图 4 所示情况,那么将前一段时间分为中断的 WCET 记为 C_{tick} 和切换代码的 WCET 两部分,其中,中断的 WCET 指的是没有任务切换时中断服务程序执行花费的最长时间,剩下部分的最大值作为切换时间.

2 软件模型

实时嵌入式系统通常包含多个完成特定工作的任务(task).每个任务包含一系列(无穷多个)的实例(task instance),任务释放执行的每个实例称为一次作业(job).根据实例的到达时间特性不同,可以将任务分为周期性任务(periodic task)、偶发性任务(sporadic task)和非周期性任务(aperiodic task)[21]:周期性任务实例的到达有固定的周期;偶发性任务实例的到达没有固定的周期,但是有最小到达时间(minimal inter-arrival time)间隔的限制,在最坏情况下可以作为周期性任务对待;非周期性任务可能随时到达,没有最小到达时间间隔的限制.固定优先级抢占式调度是实时系统里经常采用的调度,无论何时,系统中只要有更高优先级的任务就绪,便可以抢占当前运行的任务得到运行.当有多个任务就绪,调度器会选择最高优先级的任务运行.

假设一个包含 n 个任务的周期性任务集 $\tau=(\tau_1,\tau_2,\ldots,\tau_n)$,运行在采用固定优先级抢占式调度的单处理器上,任务优先级按照下标顺序从高到低排列(i 从 1 到 n),n 是最低优先级.每个任务 τ_i 表示为四元组 $\langle C_i,D_i,T_i,J_i\rangle$,其中,$C_i$ 为任务的最差执行时间,D_i 为任务的相对截止时间,T_i 为任务的周期或最小到达时间间隔,J_i 为任务的释放抖动.

到达时间 a_i^j 表示任务 τ_i 的第 j 个实例的到达时刻,那么任务的下一个实例最早在 T_i 时刻后到达,有:

$$a_i^{j+1}\geqslant a_i^j+T_i.$$

由于系统的调度器只在某些特定的时间点才会被调用,任务实例到达后可能不会立刻被释放到就绪队列,任务的释放时间 r_i^j 与到达时间 a_i^j 之间的最大时间延迟为 J_i,有:

$$r_i^j\leqslant a_i^j+J_i.$$

令 f_i^j 表示任务 τ_i 的第 j 个实例完成执行的时刻,则 τ_i 的所有作业从释放到完成执行的最大时间间隔即为任务 τ_i 的最差响应时间 R_i,有:

$$R_i=\max_j\{f_i^j-r_i^j\}.$$

任务 τ_i 的第 j 个实例的截止时间 d_i^j 是相对于其到达时间的,有:

$$d_i^j=a_i^j+D_i.$$

一个任务 τ_i 称为可调度的(schedulable),如果它的每个作业都能在截止时间之前完成,有:

$$R_i\leqslant D_i-J_i\Leftrightarrow\tau_i \text{ schedulable}.$$

任务集 τ 称为可调度的,如果其中的每个任务 $\tau_i\in\tau$ 都是可调度的.

若一个任务 τ_i 的截止时间满足 $D_i=T_i$,称 τ_i 为隐含截止时间的任务;若 $D_i\leqslant T_i$,称 τ_i 为受限截止时间的任务;若 $D_i>T_i$,称 τ_i 为任意截止时间的任务.

本文假设任务含有受限截止时间,任务之间没有相互依赖关系,定义任务的上下文切换最大时间开销为 t_{sw}.同时假定系统中包含有 m 个中断 $I=(I_1,I_2,\ldots,I_m)$,中断的优先级高于任务,中断的参数定义与任务类似,采用与任务类似的符号[22].周期性中断(如定时器中断)的触发有固定的周期,而非周期性中断(如总线通信中断)没有固定的周期,但其到达通常有一个最小的时间间隔[7,21],故最坏情况下,可当作周期性中断处理.

定义 1(关键性时刻)[19]. 任务 τ_i 的关键性时刻定义为任务 τ_i 与所有高优先级任务一齐释放的时刻.接着任务 τ_i 将经历最长的可能延迟,即它的最差响应时间.

对固定优先级抢占式调度的响应时间分析[3,4],使用下面公式:

$$R_i=C_i+\sum_{\forall j\in hp(i)}\left(\left\lceil\frac{R_i+J_j}{T_j}\right\rceil C_j\right) \tag{1}$$

其中,$\lceil\cdot\rceil$为向上取整运算,代表任务τ_i的响应期间被任务τ_j抢占的最大次数;C_i为任务τ_i的最差执行时间;J_j为高优先级任务τ_j的释放抖动;C_j为其最差执行时间;T_j为其周期,$hp(i)$代表优先级高于τ_j的任务.

我们在文献[22]中进行过分析,考虑系统中断时,某任务τ_i的响应时间为

$$R_i = C_i + \sum_{m<j<i}\left(\left\lceil\frac{R_i+J_j}{T_j}\right\rceil C_j\right) + \sum_{1\leqslant j\leqslant m}\left(\left\lceil\frac{R_i+J_j}{T_j}\right\rceil C_j\right) \tag{2}$$

由于中断优先级大于任务,用下标 1 到 m 表示中断,m+1 到 m+n 表示任务,等式右边第 2 部分为优先级大于 i 的任务对其抢占所占用的时间,第 3 部分为所有中断对其抢占所占用的时间.公式(2)迭代到其响应时间不再变化或者超过截止时间为止,得到在中断与任务嵌套下的任务的最差响应时间.

3 关键性时刻

3.1 释放抖动

如第 2 节所述,目前大部分的文献使用如下公式计算任务的最差响应时间.

$$R_i = C_i + \sum_{\forall j\in hp(i)}\left(\left\lceil\frac{R_i+J_j}{T_j}\right\rceil C_j\right),\text{判定条件为 } R_i\leqslant D_i-J_i\Leftrightarrow\tau_i \text{ schedulable.}$$

上述迭代公式中并没有包含任务τ_i的释放抖动J_i,而是将截止时间减去释放抖动作为判定条件.这意味着J_i并没有参与迭代.在实际系统中,并不一定是这种情况.

嵌入式实时操作系统一般有 4 种相关操作:开中断、关中断、开调度、关调度.其关系为:

- 开中断时,允许响应中断请求,此时若开调度,则允许任务调度;若关调度,则不允许任务调度.
- 关中断时,不允许响应中断请求和任务调度.

下面分几种情况进行讨论.

- 情况 1:在区间J_i内关中断或者关调度,则没有高优先级任务能够在J_i区间内释放,如果有高优先级任务τ_j在J_i区间到达,则其释放将延后,直到开中断且开调度为止,这段延后时间可以使用任务τ_j的释放抖动J_j来建模,公式适用.
- 情况 2:在区间 J_i 内如果开中断且开调度,则可能有高优先级任务在此期间释放,此时可以将 R_i 定义为从任务到达到完成执行的最大时间间隔,上述公式修改为

$$R_i = C_i + J_i + \sum_{\forall j\in hp(i)}\left(\left\lceil\frac{R_i+J_j}{T_j}\right\rceil C_j\right),\text{判定条件为 } R_i\leqslant D_i\Leftrightarrow\tau_i \text{ schedulable.}$$

- 情况 3:对于中断而言,由于中断的优先级高于任务,且中断到达即被释放,故在区间 J_i 内无论开关中断,计算中断开销时,都应将 J_i 进行迭代(注意:关中断不能当作中断 I_j 的释放抖动 J_j 来建模,由于中断已释放,故应当作其阻塞时间.在文献[22]中有详细说明).将中断作为高优先级任务,单列出来,计算公式修改为

$$R_i = C_i + J_i + \sum_{m<j<i}\left(\left\lceil\frac{R_i+J_j}{T_j}\right\rceil C_j\right) + \sum_{1\leqslant j\leqslant m}\left(\left\lceil\frac{R_i+J_j}{T_j}\right\rceil C_j\right),\text{判定条件为 } R_i\leqslant D_i\Leftrightarrow\tau_i \text{ schedulable} \tag{3}$$

情况 3 将在下节进行详细的讨论,情况 1、情况 2 类似,不再赘述.

3.2 中 断

本节考虑任务由系统时钟中断释放时的情形,其时间流程可以参考图 4 里的 t_1~t_7 区间,其中:t_1 时刻任务到达,t_6 时刻任务释放,t_1~t_6 即为任务的释放抖动.由上节讨论可知:这种情况下,在任务最差响应时间里包含中断开销时,需考虑系统中其他中断与时钟中断之间的关系.在文献[11]中,作者讨论了类似的问题,但是它们的研究对象是多处理器中不同的用于任务释放的中断服务程序.

如果关中断或者其他中断的优先级低于时钟中断的优先级,那么其他中断在时钟中断运行期间到达,则会被延迟到时钟中断返回,开中断后它们会立刻抢占新释放的任务得到执行;如果开中断且其他中断的优先级高于时钟中断,那么它们到达后会立刻抢占时钟中断得到运行,此时相当于增加了任务的释放抖动.无论哪种情况,都会对任务造成延迟.可见:任务与中断同时释放并不一定是其关键性时刻,需要将释放任务的时钟中断考虑在内.需要说明的是:由于中断的到达和释放之间的间隔非常短暂,所以在分析中忽略了这个时间,认为中断到达即释放.

如图 6 所示,假定时钟中断的释放时刻和任务的到达时刻为同一时刻,任务的截止时间为下一个时钟中断的起始时刻,则相对截止时间等于相邻两时钟中断起始时刻之间的时间间隔.要验证任务能否在截止时间之前完成,需要计算出从任务到达到任务完成所需要的最长时间与相对截止时间比较得出.由此,在分析时,认为任务的响应时间 R_i 为从任务 τ_i 到达到完成之间的时间,即包括了任务的释放抖动 J_i.

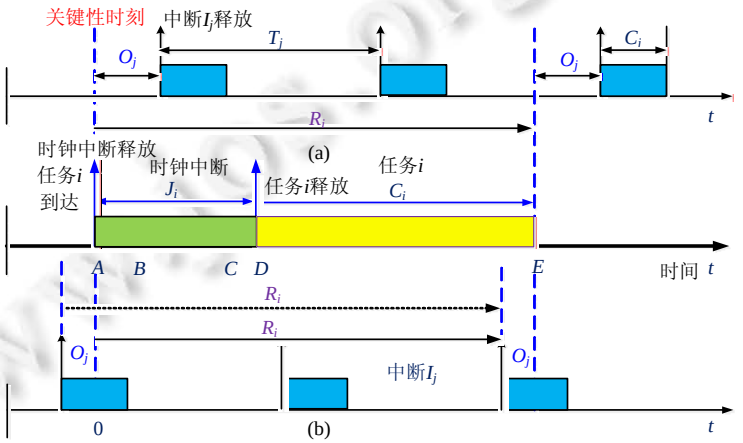

Fig.6 Critical instant including the interrupt overhead

图 6 包含中断开销时任务的关键性时刻

定理 1(任务在中断抢占下的关键性时刻). 任务在中断抢占下的关键性时刻出现在用于释放此任务的中断与其他中断一齐释放的时刻,在此之后,其他中断以其周期或最小到达时间间隔到来,任务 τ_i 将经历最长的可能延迟.

证明:假定任务 τ_i 由系统的时钟中断释放,如图 6 所示,由公式(2)易得,其他中断的周期越小,对 τ_i 可能造成的抢占次数越多,故其他中断应以其周期或最小到达时间间隔到来.假定中断 I_j 的释放时刻与时钟中断的释放时刻间有 O_j 大小的偏移,则:

- 单个中断:若 $O_j \geq 0$,如图中情况(a),易得 O_j 越小,I_j 可能的抢占次数越多.故 $O_j=0$ 时,I_j 的干扰最大.若 $O_j \leq 0$,如图中情况(b)所示,随着 O_j 的减小,R_i 随之减小,I_j 的左移并不能导致更多的抢占.故 $O_j=0$ 时,I_j 的干扰最大.可得,$O_j=0$ 为关键性时刻;
- 两个中断:考虑 I_k 和 I_j,假定二者并无依赖关系,若 $O_j \geq 0$ 且 $O_k \geq 0$,如图中情况(a),可固定 I_k 分析 I_j 或者固定 I_j 分析 I_k,与单个中断分析情况相同,易得 $O_k=O_j=0$ 时,中断 I_k 和 I_j 的干扰最大;若 $O_j \geq 0$ 且 $O_k<0$, 则 I_j 为情况(a), I_k 为情况(b),,亦可固定其一分析其二,结论不变,$O_k \geq 0$ 且 $O_j<0$ 情况类似;若 $O_k<0$ 且 $O_j<0$,二者如图中情况(b),固定 I_j 分析 I_k,此时,假设将 τ_i 的到达时刻移动到 O_j 处,可得 $O_k=O_j$ 时,等价的响应时间 R_i' 获得最大,而实际响应时间 $R_i < R_i'$,故 $O_k=O_j=0$ 干扰最大,可得,$O_k=O_j=0$ 为关键性时刻.

综上可得,对于任意数量的中断,有类似结论成立,定理得证. □

在计算任务 τ_i 在中断抢占下的最差响应时间时,可以使用公式(3),其中 $J_i=C_{tick}+t_{sw}$.

3.3 上下文切换

固定优先级抢占式调度下切换开销的简单考虑方法是在每个任务最差执行时间里加入两次上下文切换开销,即 C_j+2t_{sw}.若在高优先级任务抢占低优先级任务得到运行的过程中,有中间优先级的任务释放,则在高优先级任务运行完毕,并不会恢复低优先级任务,而是切换到中间优先级任务运行,切换开销将减少 1 次,如图 7 所示.

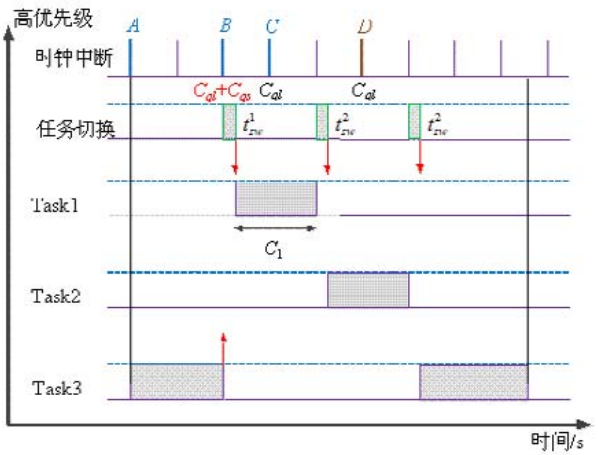

Fig.7 Preemption with scheduling overheads

图 7 包含任务调度开销的抢占

将中断处理的开销 C_{tick} 剔除,并将图 5 中的 t_1~t_2 中包含的切换开销最大值定义为 t^1_{sw},t_3~t_4 中包含的切换开销最大值定义为 t^2_{sw}.根据文献[15]中的定义,C_{ql} 是调度器将第 1 个任务从延时队列放到运行队列的开销,C_{qs} 是将多于一个任务时的其他任务从延时队列放入到运行队列的开销,则有 C_{qs}<C_{ql}.如果任务 1 和任务 2 都在点 B 释放,则任务 3 运行期间总的抢占开销是 $8C_{tick}+t^1_{sw}+2t^2_{sw}+C_{ql}+C_{qs}$.如果任务 2 在 C 点释放,则总的抢占开销为 $8C_{tick}+t^1_{sw}+2t^2_{sw}+2C_{ql}$.如果任务 2 在 D 点释放,则总的抢占开销为 $8C_{tick}+2t^1_{sw}+2t^2_{sw}+2C_{ql}$.由此说明,不但切换开销少了 1 次,而且释放开销减小.同时可以看出:如果高优先级任务之间减少重叠,容易导致更多的切换开销.

固定优先级抢占式调度的最差响应时间认为出现在理论的关键性时刻(见定义 1),即所有任务一齐释放的时刻(这里认为“t=0”时刻).然而,在考虑上下文切换开销时,这并不一定是任务的关键性时刻.观察图 8:在情况(b)中,任务 1 和任务 2 同时释放;而在情况(a)中,任务 2 和任务 1 之间有一个偏移 O_2,它大于任务 1 的 WCET.依据前述讨论,情况(a)下任务的切换次数要比情况(b)多 1 次.由于两种情况下,任务 1 和任务 2 对于任务 3 的抢占是相同的,故对于任务 3 而言,最差情况是情况(a),并非任务同时释放的情况(b).并且从图 8 可以看出:在情况(b)中,如果任务 1 运行并非其 WCET 的时间,即运行时间比 WCET 小 Δc_1,那么这同样将导致多一次切换,如果这个多的切换时间比 Δc_1 大,那么将导致任务 3 的响应时间更长,本文中不讨论这种情况.

在文献[18]中,他们得出了关于关键性时刻同样的结论,但是在计算准确的抢占次数时,他们使用了任务同时释放这一关键性时刻,与不考虑切换开销情况不同的是他们在第 1 个超周期(hyperperiod,所有任务周期的最小公倍数)里进行可调度分析来找到最差情况.然而,这样做在考虑任务间释放偏移时是不安全的.很容易从图 8 中观察到这一点:如果任务 2 的周期是任务 1 周期的整数倍,那么在同时释放的假设下,任务 2 将永远与任务 1 一同释放,故切换次数难以最大化.

很明显,与一个任务相关的最大上下文切换次数小于等于 2,如果把每个高优先级任务的 WCET 加入两个上下文切换开销,即 C_i+2t_{sw},以“t=0”时刻为关键性时刻可以计算出任务 τ_i 最差响应时间的一个上界 R_i^{UB}.类似地,如果不考虑切换开销,可以计算出任务 τ_i 最差响应时间的一个下界 R_i^{LB},那么准确地考虑开销的任务最差响应时间则位于 $(R_i^{LB},R_i^{UB}]$ 中,可以通过给每个任务分配相对于“t=0”时刻的不同的偏移来找到.

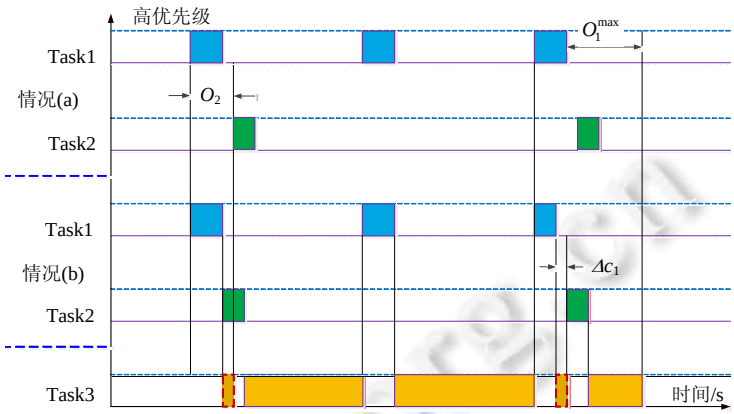

Fig.8 Task release with offset

图 8 任务释放带有偏移

4 响应时间分析

通过前述讨论观察到,可以通过给高优先级任务分配不同的偏移来找到最差情况.为了简化分析,不考虑时钟中断和上下文切换开销.保持任务 τ_i 的响应时间 R_i^{LB} 不变,高优先级任务 τ_j 的偏移 O_j 应满足某个范围 $[0,O_j^{\max})$.下面用 3 个任务的例子进行说明.如图 9 所示,情况(a)中 3 个任务在 0 时刻一齐释放,任务 1 的最后一次释放在 t_1 时刻,任务 2 的最后一次释放在 t_2 时刻,由图可知任务 3 在 t_3 时刻执行完毕,其最差响应时间为 R_3^{LB}.在情况(b)中,任务 1 有了一个偏移,它的最后一次释放将延后,结果是延迟了任务 2 的执行,却使得任务 3 后续部分可以提前执行,当偏移刚好为 $O_1^{\max}$ 时,任务 3 提前执行完毕,其响应时间小于 R_3^{LB}.

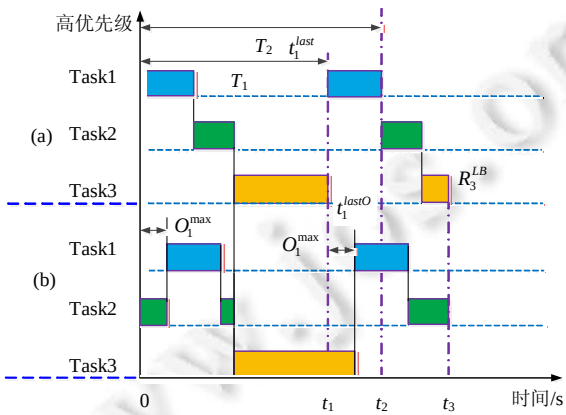

Fig.9 Example of effect of offset on response time calculation

图 9 偏移对响应时间的影响举例

为了计算 $O_j^{\max}$,使用文献[17]中的方法来建立处理器的空闲时间.令 Δ^i 代表在 R_{i+1} 期间调度任务 τ_1 到任务 τ_i 后处理器空闲时间的行向量,即 $\Delta^i=\{\Delta_{t_1}^i,\Delta_{t_2}^i,...,\Delta_{t_p}^i\}$,其中,$\Delta_t^i$ 是在 t 时刻结束的处理器空闲时间的长度,并有 $\Delta_t^i>0,t\in N$.Δ^0 定义为 $\Delta^0=\{\Delta_0^0,\Delta_{R_{i+1}}^0\}$,有 $\Delta_0^0=0$ 和 $\Delta_{R_{i+1}}^0=R_{i+1}$.$\Delta^i$ 是通过考虑在$[0,R_{i+1})$期间任务 τ_i 的所有请求后,更新 Δ^{i-1} 来构造的.令 $\Delta^{i,j-}$表示在 R_{i+1} 期间调度$\{\tau_k\},(k=1,\ldots,j-1,j+1,\ldots,i)$后处理器空闲时间的行向量,即除去任务

τ_j.事实上,Δ_t^i 实际是高优先级任务调度后,留给任务 τ_{i+1} 运行的时间,如图 10 所示.

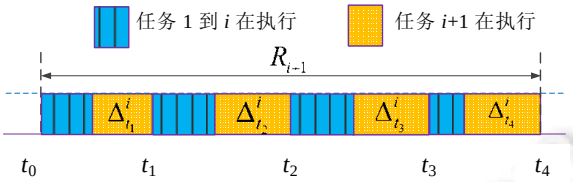

Fig.10 Illustration of Δ_t^i

图 10 Δ_t^i 的示意图

令 t_j^{last} 表示在 R_i^{LB} 期间,偏移 O_j=0 的情况下,任务 τ_j 最后一次释放的时刻,则有 $t_j^{last}=\lfloor R_i^{LB}/T_j \rfloor T_j$;令 t_j^{lastO} 表示在 R_i^{LB} 期间,偏移 $O_j=O_j^{\max}$ 的情况下,任务 τ_j 最后一次释放的时刻,则有 $t_j^{lastO}=t_j^{last}+O_j^{\max}$.通过对任务 τ_i 响应时间 R_i^{LB} 的研究,可以得出以下结论:

引理 1. 在 $[R_i^{LB}-\delta_{inst},R_i^{LB}]$ 时间段,任务 τ_i 的最后一条指令执行完毕,其中,δ_{inst} 为任务 τ_i 最后一条指令的执行时间.

证明:由 R_i^{LB} 定义可得. □

引理 2. 若任务 τ_j 从 t_j^{lastO} 处开始执行,用 $\Delta_{t_k}^{i-1,j-}$ 代表除了任务 τ_i 和任务 τ_j 外,其他任务调度后的空闲时间,则这些空闲时间在区间 $[t_j^{lastO},R_i^{LB}-\delta_{inst}]$ 内,刚好被任务 τ_j 填满.

证明:在任务 τ_i 响应期间,任务 τ_1 到 τ_i 累积的对处理器的需求一直不被满足,直到 R_i^{LB}.由 $\Delta_{t_k}^{i-1,j-}$ 的定义知,其一定被任务 τ_i 和任务 τ_j 填满.在区间 $[t_j^{lastO},R_i^{LB}-\delta_{inst}]$ 内,如果任务 τ_j 未执行完,那么 $[R_i^{LB}-\delta_{inst},R_i^{LB}]$ 区间内其必然在执行,与引理 1 矛盾;如果任务 τ_j 在 $R_i^{LB}-\delta_{inst}$ 前执行完,则 O_j 可以继续增大,直到其刚好在 $R_i^{LB}-\delta_{inst}$ 时刻执行完,而不影响任务 τ_i 的响应时间.可得在区间 $[t_j^{lastO},R_i^{LB}-\delta_{inst}]$ 内,空闲时间刚好被任务 τ_j 填满,引理得证. □

一般 δ_{inst} 时间非常短,简单起见,后续不再单独考虑其影响,令 δ_{inst}=0.利用引理 1 和引理 2,可得定理 2.

定理 2(任务的最大偏移). 不考虑上下文切换开销时,保持任务 τ_i 的最差响应时间不变,若高优先级任务 τ_j 获得最大偏移,则其他任务应以"t=0"的关键性时刻到来,且满足下述关系:

$$\begin{cases} t_j^{lastO}=t+\sum\limits_{t_k\in(t,R_i^{LB}]}\Delta_{t_k}^{i-1,j-}-C_j \\ C_j-\Delta_t^{i-1,j-}\leqslant\sum\limits_{t_k\in(t,R_i^{LB}]}\Delta_{t_k}^{i-1,j-}<C_j \end{cases} (t_j^{last}<t\leqslant R_i^{LB},\Delta_t^{i-1,j-}\in\Delta^{i-1,j-}) \tag{4}$$

证明:如图 11 所示,由引理 2 可知,若高优先级任务 τ_j 获得最大偏移,区间 $[t_j^{lastO},R_i^{LB}]$ 的空闲时间 $\Delta_{t_k}^{i-1,j-}$ 刚好被任务 τ_j 填满,则以时间 t 从 R_i^{LB} 向前搜索,累加的空闲时间等于任务 τ_j 的最差执行时间 C_j 的时刻即为 t_j^{lastO},故第 1 项成立;由于 t_j^{lastO} 时刻,任务 τ_j 开始执行,则 t_j^{lastO} 必然位于某个区间 $\Delta_t^{i-1,j-}$ 内,可得第 2 项成立,故公式(4)成立.

反之,若有公式(4)成立,将第 1 项带入到第 2 项中,有 $t-t_j^{lastO}\leqslant\Delta_t^{i-1,j-}$,故 t_j^{lastO} 位于 $\Delta_t^{i-1,j-}$ 内;由第 1 项知,t_j^{lastO} 之后累积的空闲时间刚好等于 C_j.故 t_j^{lastO} 为任务 τ_j 最大偏移时,最后一次释放后开始执行的时刻.

其次,若其他任务相对于 t=0 的关键性时刻有偏移,则空闲时间 $\Delta_{t_k}^{i-1,j-}$ 将前移,由公式(4)知,t 将减小,从而导致 t_j^{lastO} 减小,故在其他任务的偏移都为 0 时,任务 τ_j 将得到最大的偏移.定理得证. □

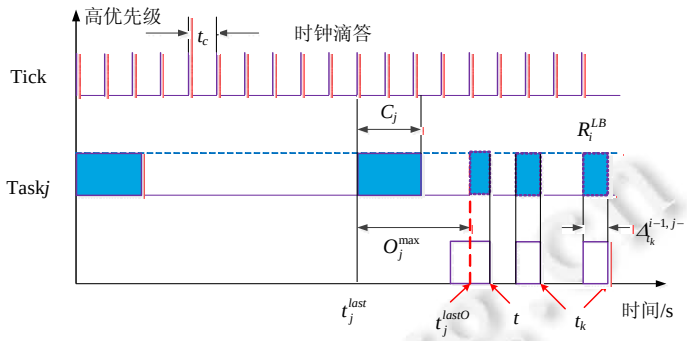

Fig.11 Maximum offset diagram

图 11 最大偏移示意图

由于加入上下文切换开销后任务响应时间会增加,故计算时假设偏移 $O_j \leqslant O_j^{\max}$,由此,可以使用文献[17]或文献[18]中的方法测试每个可能的偏移组合下各个任务的最差响应时间,也可以使用仿真的方法找到最大值.

事实上,时间是一个连续变量,其偏移的组合是无限的,所以难以做到遍历各个偏移的组合.在 RTOS 里,执行上下文切换通常是离散的时间点,称为调度点.在分析中,可以假设任务都是通过时钟中断释放的,这符合实际情况,因为这是 RTOS 用于释放周期性任务常用的方式.在这个假设下,任务仅仅在离散的时间点被释放,即周期性时钟中断执行的时间点,所以在任务 τ_i 的响应期间,高优先级任务可能的偏移组合是有限的,问题是可处理的.

将时钟滴答的大小记 t_c,任务 τ_j 相对于"t=0"的关键性时刻所有可能的释放偏移为

$$O_j = kt_c, (0 \leqslant k \leqslant \left\lfloor O_j^{\max} / t_c \right\rfloor, k \in N) \tag{5}$$

此处假定任务的偏移为时钟嘀嗒的整数倍,由于任务释放时间连续时允许的最大偏移 $O_j^{\max}$ 并不一定与时钟滴答对齐,故采用了向下取整运算 $\left\lfloor O_j^{\max} / t_c \right\rfloor$ 表示释放时间离散后最大偏移处的 k 值.同时,由于任务都在时钟滴答的整数倍处释放,故任务的周期均为时钟嘀嗒的整数倍.此种情况下,有:

$$t_j^{last} = \left\lfloor R_i^{LB} / T_j \right\rfloor T_j, t_j^{lastO} = t_j^{last} + \left\lfloor O_j^{\max} / t_c \right\rfloor t_c .$$

5 案例与仿真

5.1 中 断

本节通过真实的软硬件仿真来检验文中中断开销的分析方法和结论.仿真环境和配置与文献[22]类似,对相关细节进行了补充说明,概述如下.

系统的软硬件分别采用开源的 LEON3 处理器和 SpaceOS 操作系统,仿真环境采用 Modelsim 软件. LEON3[23]是欧洲航天局下的 Gaisler 研究所开发的使用 SPARC V8 指令集的 32 位 RISC 处理器,它的源代码由可综合的 VHDL 代码构成.基于 GPL 许可证协议,LEON3 非容错版本软核 IP 提供 VHDL 源代码.空间嵌入式操作系统 SpaceOS 是由北京控制工程研究所研制的星载计算机嵌入式操作系统,已经在多个航天型号任务上得到了应用.

整个仿真验证方案如图 12 所示:首先,通过配置 config.vhd 和 leon3mp.vhd 文件来选择 LEON3 处理器的功能特性;接着,通过配置 OSSet.h 和 OSMacro.h 来选择 SpaceOS 操作系统相应的功能特性;配置完系统运行所需的软硬件环境后,通过编写 testbench.vhd 来仿真外围的 SRAM 存储器、UART 串口和其他信号;通过编写应用程序文件 app.c 来定义系统中运行的任务程序和用户中断服务程序;最后将软硬件文件分别编译生成 Modelsim 仿真文件和用于加载到 SRAM 的可执行二进制文件.在 Modelsim 里运行仿真,得到相应的波形和程序运行时的反汇编代码,进而分析得到各个任务与中断的相应时间特性.

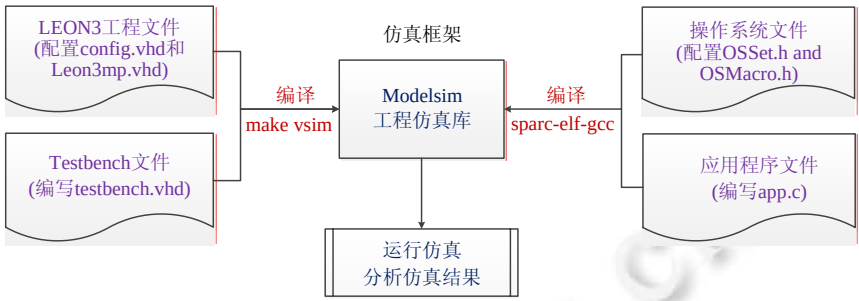

Fig.12 Simulation framework

图 12 仿真验证框架

本仿真中硬件具体配置为时钟频率为 100MHz,处理器核心数为 1,寄存器窗口数目设置为 8 个,整数单元不使用分支预测,不使用指令与数据 Cache,不开启存储器管理单元(MMU).片外连接 8M×32bit 的 SRAM 存储器,带有 5 个片内定时器和两个通用串口等.软件环境配置为使用循环调度,主周期由外部中断 0 产生,时间片中断由定时器 1 产生,任务的读写等待周期全部设置为 1(指 CPU 的指令周期数).系统另外设计为包含 4 个任务和 4 个中断,相应参数见表 1 和表 2.表中的 WCET 值是通过大量仿真后选取的最大值,表 1 中 TIMR1 括号内的值是带有任务释放时的最差执行时间.

Table 1 Interrupts of the system

表 1 系统的中断

中断I_i	优先级	释放抖动J_i (μs)	WCET C_i (μs)	周期T_i (μs)	相对截止时间D_i (μs)
TIMER1	1	0.060	22.24(97.46)	4 000	4 000
EXINT2	2	0.120	14.18	1 000	1 000
EXINT1	3	0.190	14.16	500	500
EXINT0	4	0.105	79.67	40 000	40 000
UART2	5	0.190	16.13	521.6	521.6
UART1	6	0.220	16.14	260.8	260.8

Table 2 Tasks of the system

表 2 系统的任务

任务τ_i	优先级	到达时间a_i (ms)	WCET C_i (ms)	周期T_i (ms)	相对截止时间D_i (ms)
Task1	1	0	0.003 18	40	4
Task2	2	4	12.678 72	40	24
Task3	3	28	7.188 58	40	8
Task4	4	36	0.563 79	40	4

系统的软件调度策略选择循环调度(cyclic executive)[25].循环调度是硬实时系统广泛采用的一种周期性任务调度策略,它的调度决策是静态的,离线计算好之后保存在表格中,然后在系统里以一个确定的速率被反复地执行,这个反复执行的周期称为主周期(major cycle).主周期又被分为许多小时间段,称为帧(frame)或次周期(minor cycle).每个帧都附带一个需要在帧内部执行的任务列表,通常列表里分配的任务在帧的起始时刻由系统的硬件时钟中断服务程序释放,在帧的结束时刻前必须完成.

任务 1 除了在分配的帧内执行外,每次 UART1 中断执行完毕后,还会释放任务 1 执行,由于任务 1 为最高优先级任务,故其会立刻抢占其他任务得到运行.由此造成的额外切换开销 $t_{\gamma o}$=35.87μs.最长关中断时间 $b_i^{\max}$ = 50.71μs.在具体分析的过程中,遇到以下两个需要解决的问题.

1) 某些延时操作(例如 void Delay_us())采用硬件定时器实现,这种情况下,即使被中断抢占,延时阶段的时间也不一定会被延长,故不能直接按抢占次数乘以最差执行时间计算干扰;

2) LEON3 采用 SPARC 体系结构,采用寄存器窗口机制、上溢和下溢陷阱将带入开销,需要额外分析.

5.1.1 任务延时

仿真中,任务的延时函数使用硬件定时器,在一个循环语句中反复读取定时器的值,做差与设定值进行比较来实现特定时间延时.如图 13(a)所示,函数代码在 B 点读取定时器时间初值,在 D 点读取时间终值,通过计算 B,D 两点的时间差来确定延时的时间.

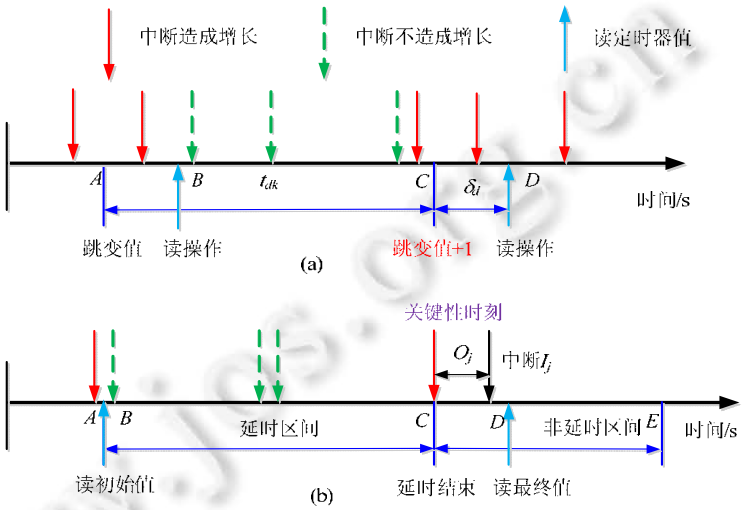

Fig.13 Analysis of delay interval

图 13 延时区间分析

一方面,由于定时器有一定的精度,其值只在离散的时间点进行更新(本仿真采用的定时器精度为 10μs),并且函数代码在连续读取时间值之间需要执行条件判断语句,故延时时间并不精确等于设定值.假设定时器的值在 A 点和 C 点发生跳变,可以看出:要想使得实际的延时时间最长,那么 B 点应当尽量靠近 A 点,D 点应当尽量远离 C 点.设期望延时时间为 t_{dk},则最坏情况下的延时时间为 $t_{dk}+\delta_d$.容易得到本仿真中 δ_d=1.5μs,分析细节便不再赘述.

另一方面,抢占发生时,真正造成延时区间增长的是那些在延时结束前到达释放并且没有返回的中断.如图 13(a)所示:B 点之前释放的中断会增加任务的响应时间;在 B 点之后到 C 点之前释放并在 C 点之前执行完毕的中断不会影响任务的响应时间;在 C 点之前释放并且到 C 点仍未执行完的中断会部分增加任务的响应时间;在 C 点之后 D 点之前释放的中断会将 D 点读取时间操作延后,也会增加任务响应时间.

综上分析,可以以 C 点为界,将任务分为延时区间和非延时区间,将非延时区间单独进行分析,假设任务执行过程中中断一直打开,有如下定理:

定理 3(非延时区间的关键性时刻). 对于延时结束后的非延时区间而言,其关键时刻出现在延时结束定时器时间值跳变时刻,所有中断一齐释放并被响应,此后所有中断各自以周期或者最小时间间隔到达.

证明:如图 13(b)所示,假定中断 I_j 的释放时间与延时结束定时器的值跳变时刻之间有 O_j 的偏移.若 $O_j\geqslant 0$,则 I_j 会增加任务响应时间,易得 O_j 越小,中断 I_j 可能的抢占次数越多,故 O_j=0 时,中断 I_j 的干扰最大.若 $O_j\leqslant 0$,则 I_j 在定时器跳变时刻之前执行部分不会增加任务响应时间.故随着 O_j 的减小,任务响应时间随之减小,中断 I_j 的左移并不能导致更多抢占.故 O_j=0 时,中断 I_j 的干扰最大.可得,O_j=0 为关键性时刻.对于任意数量的中断,有类似结论,定理得证. □

设任务 3 的程序中延时操作个数为 n_d,每个延时操作的时间为 $d_{i,k}$,则被延时造作分割成的区间个数为 n_d+1 个,其最差执行时间分别为 $C_{i,k}$.分别以 $C_{ik}+\delta_d$ 为初值,利用公式(2)计算任务被分割成的非延时区间的最差响应时间 $R_{i,k}$ 的值.根据前面分析,得到任务的最差响应时间为

$$R_i = \sum_{k=1}^{n_d+1} R_{i,k} + \sum_{k=1}^{n_d} d_{i,k} \tag{6}$$

这种方法的缺点是:当两个相邻延时区间间隔较短时,会出现较大的过估.见表 3,任务 3 中有 3 个延时,其中,第 1 个延时只有 10μs,其与第 2 个延时的间隔最大只有 2.97μs,相对于中断的周期要小得多,这样如果单独考虑,定义关键时刻进行计算,则会产生较大过估.在分析中,将其当作非延时区间对待.

Table3 Composition of Task3

表 3 任务 3 的组成

名称	WCET (μs)	名称	延时时间(μs)
$C_{3,1}^d$	18.52	$d_{3,1}$	10
$C_{3,2}^d$	2.97	$d_{3,2}$	2 000
$C_{3,3}^d$	70.5	$d_{3,3}$	5 000
$C_{3,4}^d$	95.91	…	…

5.1.2 上下溢陷阱

本仿真验证采用我们在文献[24]中提出的方法对寄存器窗口溢出陷阱的开销进行分析,方法的内容不再赘述,只对一些情况进行说明(在本例中,上下溢陷阱服务程序的最差执行时间分别为上溢陷阱 t_{of}=2.67μs,下溢陷阱 t_{uf}=3.47μs).

1) 抢占出现时,被抢占任务使用过的所有寄存器窗口都要保存,而且保存的寄存器个数越多,所花费时间越长,那么最长切换时间应出现在任务调用深度最大时.这个切换时间对应图 5 中的 t_1~t_2.若被抢占任务是由中断释放的,如此例中的中断 UART1 每次都会释放任务 1,那么这个切换时间对应图 4 中的 t_5~t_6.在计算时,安全简便的处理方式是这个切换时间全部取成任务调用深度最大时的值;

2) 抢占返回时,被抢占任务只恢复一个新窗口,任务恢复运行后每次执行 RESTORE 指令返回都会发生一次下溢,同样在任务调用深度最大时,需要返回的次数最多,那么发生下溢的次数也越多,所花费时间越长.而且文献[24]指出,每次返回潜在的可能发生两次下溢陷阱.下面以任务 4 为例分析这种情况.

图 14 为任务 4 的函数调用图.先计算出任务 4 响应期间的最大上下文切换次数为 n_c=6 次.为了得到最差情况,这些切换应该首先发生在最深层,即 1 次在 Func4 或 Func5,1 次在 Func7 或 Func8,1 次在 Func11 到 Func16.这 3 次切换在 Func3,Func6,Func10,Func1 和 Task4 返回时同样可以引起下溢陷阱.剩下的 3 次切换每个可以引起一次下溢陷阱,则总的下溢次数为 3+3+5=11.对上溢而言,当切换发生时,如果下一个运行的任务是新运行任务(未被抢占),那么将有 n_r−2=6 个寄存器窗口;如果是恢复运行的任务(前面被抢占),那么将有 n_r−1=7 个寄存器窗口(没有被 RTOS 占用的窗口).在两次切换之间,利用无任务切换时的计算公式计算最大上溢次数.

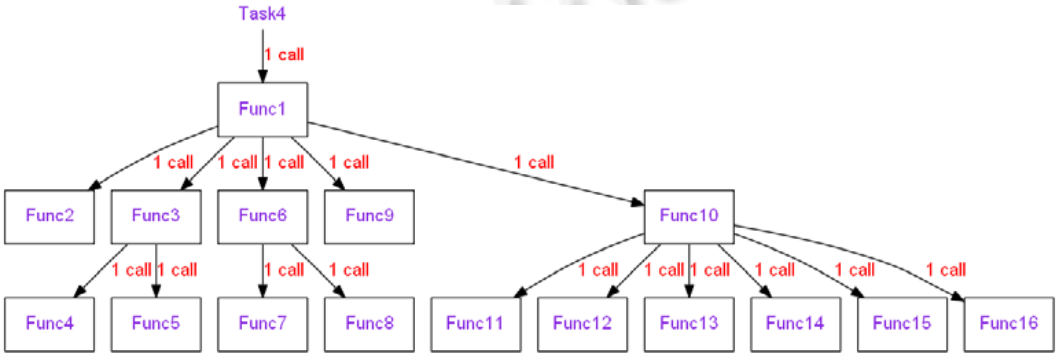

Fig.14 Call graph of task4

图 14 任务 4 的函数调用图

具体到本例中,任务 2、任务 3 和任务 4 发生下溢最大次数分别为任务 2 是 4 次、任务 3 是 10 次、任务 4

为 11 次.由于频繁的任务切换,没有上溢陷阱.

5.1.3 仿真结果

综上所述,计算得到中断影响下的任务最差响应时间与仿真得到的最差响应时间对比见表 4.仿真使用前述配置并且使用不同的中断偏移重复多次,仿真值一栏给出的是所测时间的最大值.任务 3 的过估最小,原因是其延时区间时间较长,非延时程序运行时间较短,所以分析结果比较准确,同时也说明对延时区间的分析方法比较有效.任务 4 的过估最大,主要是因为其执行时间最短,但是函数调用关系和控制流复杂,测试很难覆盖到最差情况,同时分析也有较大过估.可以看出,最大过估仅为 13.25%,说明本文所述的中断开销的计算方法是非常有效的.

另外,通过本例的分析过程可以看出:在对实际系统进行分析时,会遇到很多实际的困难,所以在对理论方法进行研究的同时,也需要进行对方法的应用进行探索.本文提出的仿真架构(如图 12 所示),为应用验证提供一个很好的平台.

Table 4 Analysis and simulation results

表 4 分析和仿真结果

名称	分析值(ms)	仿真值(ms)	过估率(%)
Task2	23.808 10	22.764 64	4.58
Task3	7.770 85	7.502 10	3.58
Task4	1.323 95	1.191 44	13.25

5.2 上下文切换

由于任务上下文切换的机理比较明确,而使用实际系统进行仿真测试过于耗时,故本节使用仿真工具进行分析.我们用一个例子来阐述第 4 节提出的响应时间计算方法.假设系统共有 4 个任务$\{\tau_1,\tau_2,\tau_3,\tau_4\}$,它们的参数见表 5.假设时钟滴答 t_c 为 0.5,上下文切换的时间上限 t_{sw} 为 0.05.

Table 5 Task parameters

表 5 任务的参数

名称	优先级	WCET C_i	周期T_i
τ_1	1	1	6
τ_2	2	2	8
τ_3	3	3	12
τ_4	4	4	24

使用开源的可调度性分析工具集 MAST[26]来做分析.JSimMAST 是其中的一个事件触发的仿真工具,用它来仿真每种偏移组合下任务的最差响应时间.以寻找任务 τ_4 的最差响应时间为例,分析过程如下.

- 首先,仿真不考虑上下文切换开销时,在 t=0 时刻所有任务一齐释放的情形,得到 $R_4^{LB}=20$,并且可以得到 $\Delta^{3,1-}=\{\Delta_8^{3,1-}=3,\Delta_{12}^{3,1-}=2,\Delta_{16}^{3,1-}=1,\Delta_{20}^{3,1-}=2\}$.使用公式(4)计算出 $t_1^{last}=18,t_1^O=19$ 和 $O_1^{\max}=1$.接着,使用公式(5)可以计算出 $0\leqslant k<2$,可得 O_1=0,0.5,1.类似的,可以计算出 O_2=0,0.5,1 和 O_3=0,0.5,1,1.5,2.当然,也可以使用仿真的方法来找到最大偏移,只要令其他任务的偏移都为 0,在 $R_i-t_j^{last}-C_j$ 范围内测试 O_j 即可;
- 其次,使用不同的任务偏移组合来仿真任务集.由于只关注任务 τ_4,故只给出它的最差响应时间仿真结果为 20.9.而所有任务一齐释放时的最差响应时间为 20.75.可以观察到,任务 τ_4 响应时间的增长允许它可以接受高优先级任务更大的偏移;
- 第三,增大偏移的数值,令 O_1=1.5,O_2=1.5 和 O_3=2.5,并再次执行仿真过程.现在观察到某些情况下,R_4 小于 20.例如,O_1=0,O_2=0 和 O_3=2.5 情况下,有 R_4=14.5;而 O_1=0.5,O_2=0 和 O_3=2.5 情况下,有 R_4=20.85.原因是后者带入了更多的上下文切换开销,从而增大了 R_4;
- 最后得到任务 τ_4 的最差响应时间的准确值为 20.95.这个结果实际上与给每个高优先级任务的 WCET 增加两次切换开销的分析结果相同,仅仅是任务 τ_4 结束时,最后一次切换开销在仿真中没有计算而已.

更进一步,如果假设时钟滴答 t_c 为 1,那么有 O_1=0,1,O_2=0,1 和 O_3=0,1,2,从而得到 R_4=20.85.如果假设 t_c 是 2,

那么有 O_1=0,O_2=0 和 O_3=0,2,从而得到 R_4=20.8.这些情况下,利用本文方法计算得到的最差响应时间将更加精确.同时也可以看出:上下文切换次数与时钟滴答的大小是密切相关的,某些时候,增大时钟滴答可以减少切换次数,但可能使系统的响应变迟缓.

6 总 结

响应时间分析方法多基于系统任务的抽象模型,将其应用于实际系统分析时,还需要考虑很多实际的因素,如中断、上下文切换、延时区间、寄存器窗口溢出陷阱等.目前对于这些实际因素的研究较少,这在一定程度上阻碍了其在一些安全关键系统中的实际应用.

本文对响应时间分析方法应用于实际系统进行了有益的探索,对实时嵌入式系统里的中断和上下文切换时序进行了详细的讨论,给出了这些开销的估计方法,并分别通过真实的硬件进行了仿真.同时,在理论方面,本文对任务由中断释放、考虑上下文切换开销、程序包含定时器延时等情况下的任务的关键性时刻进行了讨论,并给出了相关证明.本文仍有一些不足之处,如对于阻塞和 cache 相关的抢占延迟等因素以及多处理器系统缺乏讨论,这将作为未来的工作.

References:

[1] Liu JWS. Real-Time Systems. Prentice Hall, 2000.

[2] Yang MF, Gu B, Guo XY, Dong XG, Wang Z, Chen R. Aerospace embedded software dependability guarantee technology and application. Scientia Sinica Technologica, 2015,45(1):198−203 (in Chinese with English abstract). [doi: 10.1360/N092014-00485]

[3] Joseph M, Pandya P. Finding response times in a real-time systems. The Computer Journal, 1986,29(5):390−395.

[4] Audsley NC, 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.

[5] Regehr J. Safe and structured use of interrupts in real-time and embedded software. In: Lee I, Leung JYT, Son SH, eds. Handbook of Real-Time and Embedded Systems. Chapman and Hall/CRC Press, 2007. [doi: 10.1201/978 1420011746.ch16]

[6] Brylow D, Palsberg J. Deadline analysis of interrupt-driven software. IEEE Trans. on Software Engineering, 2004,30(10):634−655. [doi: 10.1109/TSE.2004.64]

[7] Jonathan K, Dorsa S, Sanjit AS. Timing analysis of interrupt-driven programs under context bounds. In: Proc. of the Formal Methods in Computer-Aided Design (FMCAD). 2011.

[8] Leyva-del-Foyo LE, Mejia-Alvarez P, Niz D. Integrated task and interrupt management for real-time systems. ACM Trans. on Embedded Computing Systems, 2012,11(2):1−31. [doi: 10.1145/2220336.2220344]

[9] Jeffay K, Stone DL. Accounting for interrupt handling costs in dynamic priority task sysetms. In: Proc. of the IEEE Real-Time Systems Symp. 1993. 212−221.

[10] Sandström K, Eriksson C, Fohler G. Handling interrupts with static scheduling in an automotive vehicle control system. In: Proc. of the IEEE Int'l Conf. on Real-Time Systems and Applications (RTAS). 1998.

[11] Brandenburg BB, Leontyev H, Anderson JH. An overview of interrupt accounting techniques for multiprocessor real-time systems. Journal of Systems Architecture, 2011,57:638−654. [doi: 10.1016/j.sysarc.2010.05.011]

[12] Cofer D, Rangarajan M. Formal verification of overhead accounting in an avionics RTOS. In: Proc. of the IEEE Real-Time Systems Symp. 2002. 181−190.[doi: 10.1109/REAL.2002.1181573]

[13] Bimbard F, George L. FP/FIFO feasibility conditions with kernel overheads for periodic tasks on an event driven OSEK system. In: Proc. of the IEEE Int'l Symp. on Object and Component-Oriented Real-Time Distributed Computing. 2006. [doi: 10.1109/ISORC.2006]

[14] Katcher DI, Arakawa H, Strosnider JK. Engineering and analysis of fixed priority schedulers. IEEE Trans. on Software Engineering, 1993,19(9):920−934. [doi: 10.1109/32.241774]

[15] Burns A, Tindell K, Wellings A. Effective analysis for engineering real-time fixed priority schedulers. IEEE Trans. on Software Engineering, 1995,21(5):475−480.

[16] Gabriëls R, Gerrits D. Accounting for overhead in fixed priority pre-emptive scheduling. Department of Mathematics & Computer Science, Technische Universiteit Eindhoven, 2007.

[17] Echagüe J, Ripoll I, Crespo A. Hard real-time preemptively scheduling with high context switch cost. In: Proc. of the Euromicro Workshop on Real-Time Systems (ECRTS). 1995. [doi: 10.1109/EMWRTS.1995.514310]

[18] Yomsi PM, Sorel Y. Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. In: Proc. of the Euromicro Conf. on Real-Time Systems (ECRTS). 2007.

[19] Liu CL, Layland JW. Scheduling algorithms for mul-tiprogramming in a real-time environment. Journal of the ACM, 1973,20(1): 46−61.

[20] Bryant RE, O'Hallaron DR. Computer Systems: A Programmer's Perspective. 2nd ed., Prentice Hall, 2011.

[21] Buttazzo GC. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. 3rd ed., Boston: Springer-Verlag, 2011. [doi: 10.1007/b102312]

[22] Yu GL, Yang MF, Xu J, Jiang H. Worst case response time analysis of multilevel interrupt systems. Chinese Space Science and Technology, 2016,36(2):28−36 (in Chinese with English abstract). [doi: 10.16708/j.cnki.1000-758X.2016.0003]

[23] LEON3. http://www.gaisler.com/

[24] Yu GL, Yang MF. Timing analysis of register windows traps for real time system based on SPARC. Aerospace Control, 2015,33(6): 70−75 (in Chinese with English abstract). [doi: 1006-3242(2015)06-0070-06]

[25] Baker TP, Shaw A. The cyclic executive model and ada. In: Proc. of the IEEE Real-Time Systems Symp. (RTSS'88). 1988.

[26] MAST. http://mast.unican.es/

附中文参考文献:

[2] 杨孟飞,顾斌,郭向英,董晓刚,王政,陈睿.航天嵌入式软件可信保障技术及应用研究.中国科学:技术科学,2015,45(2):198−203.

[22] 于广良,杨孟飞,徐建,姜宏.面向多级中断系统的任务最差响应时间分析.中国空间科学技术,2016,36(2):28−36.

[24] 于广良,杨孟飞.基于 SPARC 的实时系统寄存器窗口溢出时间分析.航天控制,2015,33(6):70−75

于广良(1986－),男,山东莱州人,博士,工程师,主要研究领域为实时系统,嵌入式系统,可信软件.

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设计,控制计算机,可信软件.