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.

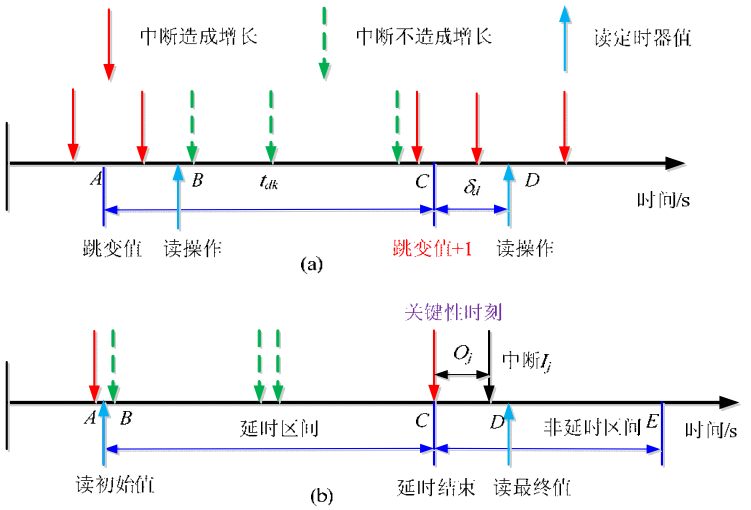

Fig.13 Analysis of delay interval

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.

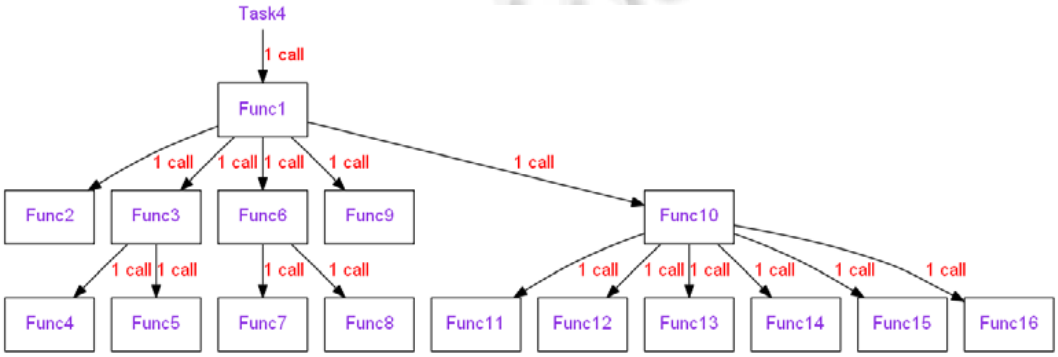

Fig.14 Call graph of task4

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.