文章编号:1005－0523(2005)04－0110－03

嵌入式实时操作系统的调度策略

周 洁,李正凡

(华东交通大学 信息工程学院,江西 南昌 330013)

摘要:在资源有限的嵌入式系统中,任务调度器的好坏很大程度上决定了系统的性能.本文分析了实时系统中有代表性的静态以及动态调度算法,并总结了各自的优缺点.在此基础之上,对嵌入式实时内核 μc/os－II 的调度算法进行了优化.

关 键 词:嵌入式实时操作系统;多任务优先级;调度

中图分类号:TM392.3 **文献标识码**:A

0 引 言

嵌入式系统与通用计算机系统不同,应用程序可以没有操作系统而直接在芯片上运行,但是为了合理地调度多任务、利用系统资源,系统一般以成熟的实时操作系统(RTOS)作为开发平台.实时操作系统的首要任务是调动一切可利用的资源完成实时控制任务,其次才着眼于提高计算机系统的工作效率.实时并非指"快速",实时的含义是在规定的时间内能够传递正确地结果.实时系统按严格的时间限制,可划分为硬实时和软实时系统.前者如果在不满足响应时限,响应不及时或者反应过早的情况下会造成灾难性的后果,而后者只会导致系统性能的退化.

调度是指在有限的处理单元上对具有某些已知特征的任务集执行顺序的设计.在多任务嵌入式系统中,单纯通过提高处理器速度无法保证对重要性各不相同的任务进行合理地调度.这种任务调度只能由优化编写的系统软件来完成.在设计和实现嵌入式实时软件过程中,抢占多任务是一个普遍使用的结构,然而抢占多任务带来的代价包括 CPU 带宽的浪费以及增加的内存总开销.当一个任务退出运行时,实时操作系统需要保存它的运行现场信息、插入相应的队列、并依据一定的调度算法重新选择一个任务使之投入运行.调度算法实际上就是系统所采取的调度策略.大多数实时内核都采用基于优先级的调度算法.本文主要讨论与 CPU 相关的调度原理和算法,并给出了一种在优先级个数有限的情况下优先级分配的有效方法.

1 静态调度

静态调度是在系统开始运行前进行调度的,严格的静态调度在系统运行时无法对任务进行重新调度.静态调度的目标是把任务分配到各个处理机,并对每一处理机给出所要运行任务的静态运行顺序.静态调度算法实现简单,调度的额外开销小,在系统超载时可预测性好.但也具有很大的局限性,例如资源利用率低、受系统支持的优先级个数限制以及灵活性和自适应性差等.下面介绍两种常见的静态调度算法.

1.1 速率单调调度(Rate Monotonic Scheduling)

RMS 算法将最高优先级赋予最高执行频率的任务,以单调的顺序对剩余的任务分配优先级.由于采用抢占式的调度方式,高优先级的任务就绪后

收稿日期:2004－12－18

作者简介:周 洁(1979－),女,江西南昌人,华东交通大学助教,在读研究生.

立即抢占正在运行的任务.Liu 和 Layland 的理论证明了下列公式,即用 RMS 调度的独立的周期性任务总能满足其截止时间的要求.

$C_1/T_1+C_2/T_2+\cdots+C_i/T_i+\cdots+C_n/T_n\leqslant U(n)=n(2^n-1)$

式中:C_i 为任务 i 的最坏执行时间;T_i 为任务 i 的周期.

RMS 算法的优点是开销小,灵活性好,可调度性测试简单.在任务的截至时间等于其周期的条件下,速率单调算法已被证明是静态最优的调度算法[1].

1.2　截止时间单调调度(Deadline Monotonic Scheduling)

DMS 是在速率单调调度的基础上发展起来的.不同的是任务的优先级按截止时间来分配.截止时间短的任务优先级高,截止时间长的任务优先级低.

截止时间调度具有与速率单调算法相同的优点.但 DMS 算法放松了对任务的周期必须等于其截止时间的限制.在任务的截止时间小于或等于其周期的情况下,DMS 已被证明是静态最优的调度算法.

2　动态调度

在嵌入式实时系统中动态调度依赖于任务的优先级.优先级可以静态分配或者依据不同的特征参数,如截止时间、空闲时间或关键性(即任务的重要程度或者价值)等进行动态分配.动态调度可以是抢占式的或非抢占式的.前者用优化的标准动态调动任务,从而减少因超过截止时间而失败的任务数,后者检查系统中每一个任务,看是否有新任务可以加入调度列表,从而满足其截止时间的要求.非抢占式内核的优点是易于分析、实现,利于保存额外的上下文切换,节省抢占式内核中由于采用互斥机制带来的开销,其缺点是可能使某些任务不能满足其截止时间的要求.几个比较著名的商用嵌入式实时操作系统,如 QNX 、VxWorks 提供的都是"抢占式任务调度".以下介绍的是两个著名的动态调度算法 EDF 和 LSF.

2.1　最早截止时间优先算法(Earliest Deadline First)

抢占式 EDF 调度算法是一个动态优先级驱动的调度算法,其中分配给每个任务的优先级根据它们当前对最终期限的要求而定.当前请求的最终期限最近的任务具有最高的优先级,而请求最终期限最远的任务被分配最低优先级.这个算法能够保证在出现某个任务的最终期限不能满足之前,不存在处理器的空闲时间[2].该算法基于以下假设:

1) 任何任务不存在不可抢占的部分,且抢占的代价可以忽略;

2) 只有处理器请求是有意义的,内存、I/O 和其它资源请求可以忽略;

3) 所有的任务都是无关的,不存在先后次序的约束;

4) 任务的相对最终期限与它的周期相等.

因此抢占式 EDF 调度算法对于给定周期性任务集可调度性的充分必要条件为:

$$\sum_{i=1}^{n}\frac{e_i}{p_i}\leqslant 1$$

其中,e_i,p_i 分别为任务集中任务 i ($1\leqslant i\leqslant n$)的执行时间和周期.由此可知,抢占式 EDF 调度算法最大的优势在于,对于任何给定的任务集,只要处理器的利用率不超过 100%,就能够保证它的可调度性.EDF 算法在系统的负载相对较低时非常有效.但在系统负载较重时,系统性能急剧下降,引起大量任务错过截止期,甚至可能导致 CPU 时间大量花费在调度上,在这时系统的性能还不如 FIFO (First In First Out)方法.

2.2　最小松弛时间算法(Least Slack Time First)

LSF 算法计算任务的松弛时间,其中松弛时间为任务截止时间与剩余执行时间之差.该算法在任意时刻把最高优先级分配给具有最小松弛时间的任务,以此来保证紧急任务的优先执行.然而,由于等待任务的松弛时间是严格递减的,其等待执行的缓急程度也随时间越来越紧迫,因此在系统执行过程中,等待任务随时可能会抢占当前执行的任务.LSF 算法造成任务之间的频繁切换或称为颠簸(thrashing)现象较为严重.颠簸现象增大了系统开销,并限制了 LSF 算法的应用.

3　静态和动态调度算法的选择

嵌入式实时系统中资源是非常有限的,所以开销要尽可能小.开销主要包括运行开销和调度开销.运行开销与队列分析和从调度队列中增加、删除任务相关.每个任务在一个调度周期内至少被阻塞和唤醒一次,所以任务调度器在一个周期内不得

不对一个任务进行两次选择.静态调度对时间触发系统的设计很适合,但它必需事先进行设计,综合考虑选择众多参数,通常需要过大范围地设计以能够处理最不可能的事件.动态调度适合设计事件触发的系统,在执行期间能够动态做出决定,并且它在资源利用方面比静态调度有更大的潜力.结合上面提及的几个著名算法进行分析、比较.如 RMS 算法,它根据任务的执行频率设置优先级,有较小的运行开销,但执行频率最高的任务不一定是最重要的.而像 EDF 和 LSF 这样的动态调度策略需要较高的调度开销来在运行时对操作进行评估.因为抢占发生时,运行上下文的保存和恢复及相应的中断等额外开销大于任务正常结束,启动另一个任务的额外开销.此外,动态调度策略在调度量过大时对那些错过最终期限的操作不提供有效控制手段.当一个操作被加入调度队列以获得更高利用率时,所有操作的实时安全性就降低了.因此对所有操作来说,当系统过载时,操作错过截止期的几率增加了.通过上述分析,不难看出无论采用何种调度策略,都是各有优点和弊端的.

4 对实时内核 μc/os—II 任务调度算法的改进

μc/os—II 是目前流行的一个开源免费的实时内核.μc/os—II 2.62 可以管理 64 个任务,保留了其中 8 个给系统,所以应用程序最多可以有 56 个任务.并且赋予每个任务的优先级必须是不相同的[3].这样的限制虽然简化了任务管理系统,但也给大型应用带来了很大的不便.通过上文对静态和动态调度策略的分析、比较,可以结合 RMS 算法具有较小运行开销和 EDF 算法具有较小调度开销的特点,得到一个改进的优先级分配算法,即在多个任务要求调度时,选择截止时间最短(即优先级最高)的任务运行;若任务优先级相同,则选择运行频率最高的任务运行.从而突破原系统对任务数量的限制,并且去除对每个任务必须有不同优先级的要求.

算法改进的关键之处是对优先级相同的任务建立散列表.在散列表中按照任务执行频率进行降序排列,这样就能保证该优先级散列表中的第一个任务就是在此优先级上执行频率最高的任务.任务管理系统调用接口包括任务创建、任务删除、更改任务优先级、挂起任务、恢复任务、与任务堆栈操作相关的系统接口和其他一些接口函数等.本文只简要说明改进后任务创建和任务调度的算法,伪码如下.

```
OSTaskCreate
{ 分配给任务的优先级合法性检查;
建立任务控制块;
建立任务堆栈 OSTaskStkInit();
if(在该优先级上已有任务存在)
{ 比较任务的执行频率;
将该 TCB 插入到散列表中(降序);
}else //该任务是散列表中的第一个任务
{ 插入到已用任务控制块链表 OSTCBList 中;
}
}
OSSched
{ if ((OSLockNesting | OSIntNesting) == =0)
查找就绪状态下优先级最高的任务;
if(此优先级>当前运行任务的优先级)
if(此优先级任务数>1)
{ 在该优先级散列表中查找执行频率最高的任务;
任务切换;
}else{ 任务切换;} //该优先级上只有一个任务
}
```

5 结束语

通过对实时系统中的静态以及动态调度算法进行分析、比较,以嵌入式实时内核 μc/os—II 为例,结合 RMS 算法具有较小运行开销和 EDF 算法具有较小调度开销的特点,对 μc/os—II 的任务调度策略进行了改进,是对嵌入式系统调度问题的一次有益探索.当然,改进的调度算法将实际的操作环境作了简化,只考虑了周期性任务组成的实时系统,在一定程度上限制了他们在工程实际中的应用.将非周期性的任务也加以全面考虑,是本文下一步的研究工作.

(下转第 133 页)

Simple Intelligent Dynamoelectrical Car

JIANG Zhi-ling

(School of Electrical and Electronic Engineering, East China Jiaotong University, Nanchang 330013, China)

Abstract: The designed car, which is controlled by 16-bit MCU-SPCE061A under combining the infrared sensor, ray sensor, metal detector, photoelectric switch, quickly switch sensor, is a kind of intelligent car. The main control system, which chooses lingyang's singlechip of 16-bit SPCE061A as CPU, not only operates quickly, but also has functions such as simple configuration, electromotor's exact control, steadily advance, sound realplay.

Key words: intelligent car; SPCE061A; sensor

(上接第 112 页)

参考文献:

[1] Liu C L, Layland J W. Scheduling algorithm for multiprogramming in a hard real-time environment[J]. Journal of the ACM, 1973, 20(1): 40—61.

[2] Krishna C M, Shin G K. Real-Time System[M]. Columbus, OH: McGraw-Hill Companies, Inc. 1997.

[3] Jean J. Labrosse. μC/OS-II The Real-Time Kernel [M]. 北京:北京航空航天大学出版社, 2003.

Scheduling Strategy of Embedded Real-time Operation System

ZHOU Jie, LI Zheng-fan

(School of Information Engineering, East China Jiaotong University, Nanchang 330013, China)

Abstract: Task scheduler largely determine performance of the embedded system with limited resources. This paper analyses typical static scheduling algorithms and dynamic scheduling algorithms, and summarizes their merits and faults. This paper also optimizes scheduling algorithm of the embedded real-time kernel μc/os-II based on these.

Key words: embedded real-time operation system; multitask; priority; scheduling