第 36 卷 第 4 期 计 算 机 工 程 2010 年 2 月
Vol.36 No.4 Computer Engineering February 2010
·工程应用技术与实现· 文章编号：1000—3428(2010)04—0233—03 文献标识码：A 中图分类号：TP316

基于 OSEK 的任务调度算法改进及实现

毛成勇，高慧敏
(太原科技大学系统仿真与计算机应用研究所，太原 030024)

摘 要：针对嵌入式操作系统中的任务调度算法不能保证临近时限的低优先级任务得以运行的情况，提出一种任务管理机制和任务调度算法。该算法通过一种链表结构来解决任务调度时系统开销大的问题，并在任务建立时确定任务的时限。当任务的时限临近时提高其优先级，使其得以运行，使系统在保证高优先级任务得以运行的同时，确保低优先级任务得以运行，从而保证了系统的实时性。
关键词：标准；任务管理；任务调度；操作系统

Improvement and Realization of Task Schedule Algorithm Based on OSEK

MAO Cheng-yong, GAO Hui-min
(Division of System Simulation and Computer Application, Taiyuan University of Science and Technology, Taiyuan 030024)

【Abstract】Aiming at the problem that the task schedule algorithm in the embedded operating system can not guarantee the low priority task run in time. This paper proposes a kind of task management mechanism and the task schedule algorithm, which set up a list structure. This structure can avoid overhead when the task schedule, and establish the time limit of the task when it is established. When the time limit of the task approaches, its priority will be enhanced, so the task can run successfully. Using the algorithm in this paper, the high priority task of the system can run, meanwhile the low priority task can run in time, and so the real-time of system can be guaranteed.
【Key words】standard; task management; task schedule; operating system

OSEK/VDX 规范集[1]是由德国汽车工业界于 1993 年联合推出的汽车电子开放式系统与接口 OSEK 及由法国汽车工业界的相似规范 VDX 合并而成。嵌入式操作系统中的任务调度算法从简单的合理安排任务循环，发展到基于优先级的速率单调调度(RMS)[2]、截止时间单调调度(DMS)、最早时限优先(EDF)、最小松弛时间(LSF)[3]等算法。本文提出一种新的任务调度算法——差分时限优先任务调度算法，该算法在保证高优先级任务得以运行的同时，也保证那些达到时限的低优先级的任务能够得以运行。

1 任务调度算法的改进

1.1 不同优先级任务间的调度改进

1.1.1 EDF 算法成立的条件

对于不同优先级任务之间的调度，采用单处理器最优算法——EDF 任务调度算法。在 EDF 任务调度算法中，系统根据各任务距离其绝对时限的接近程度给任务分配优先级，任务的截止期越早，其优先级越高，反之越低。

最早时限优先任务调度算法理论上有 3 个假设条件[3]：

(1)任务总可以被抢占，抢占的代价可以忽略不计；

(2)只考虑任务的处理器需求，内存、I/O 和其他资源请求忽略不计；

(3)任务间相互独立，不存在先后关系。

基于上述假设，假定一个实时系统中有 n 个任务，第 i 个任务用 T_i 标识。对应于每个任务的任务周期为 t_i，运行时间为 R_i，时限为 D_i。忽略任务的切换带来的 CPU 消耗。该系统可以使用 EDF 算法调度的充分必要条件是：

$$\sum_i^n \frac{R_i}{t_i} \le 1 \quad (1)$$

对于非周期任务亦是如此，非周期任务大多是随机事件，其任务 T_i 发生的时间间隔在区间[0 t_i]上服从于参数为(λ, t_i)的泊松分布，λ 为单位时间内发生次数的期望，则非周期任务可以看作平均周期为 $1/\lambda$ 的周期任务进行分析。同样，可以利用式(1)进行可调度性判断。

实际情况下，任务的调度对 CPU 所带来的负担不可忽略，假设任务调度消耗 CPU 时间为 Δ_i，则式(1)变为

$$\sum_i^n \frac{R_i + \Delta_i}{t_i} \le 1 \quad (2)$$

可见，实时系统对调度算法的要求是：在保证系统中所有任务在其时限内得到处理的前提下，尽可能地减小 Δ_i。

实际系统总会有一些优先级较高，能够在时限内被处理完或者没有时限要求的任务。在这种情况下，给这些任务运用最早时限优先任务调度算法是没有必要的，徒增系统负担。

1.1.2 任务调度算法的改进

在本文提出的算法中，对一个任务，首先尽可能通过单纯的基于优先级的任务调度算法来调度执行，当其执行时间接近时限时，采取最早时限优先任务调度算法进行调度，从而使任务能在时限内得以执行。

该算法需要在调用调度程序时，逐一为每个任务进行时

作者简介：毛成勇(1978－)，男，硕士研究生，主研方向：嵌入式系统；高慧敏，教授、博士
收稿日期：2009-07-19 **E-mail：**maochengyong0624@163.com

限更新和检测将耗费较长时间，系统中任务数越多，所需时间越长；而且调度程序不能被抢占，该段代码要以临界资源处理，需提前关中断。因此，任务调度时巨大的时间开销是实时系统所不能容忍的，严重影响了系统的响应。

本文通过链表结构(差分时限链)来解决此问题。通过差分时限链来更新、检测每个任务的时限。其每一个元素均为结构体变量，结构体成员由前向指针、后向指针和时限 3 个变量组成。

差分时限链中任务时限值和插入位置的确定。当第 $i(i\geq 0)$ 个任务建立时给定任务的时限为 D_i，差分时限链中该任务的时限为 d_i，其大小由式(3)确定，其中 $d_j^{'}(0\leq j<n)$为差分时限链中第 j 个元素所指任务在时限链中的时限值，n 为差分时限链的长度。插入的位置，应插入在差分时限链中第 j 个元素所指任务的后边。

$$d_i = D_i - d_0^{'} - d_1^{'} - \cdots - d_j^{'} > 0 \tag{3}$$

如果插入的元素不是在差分时限链的最后，则新插入元素后边各元素的值应为

$$d_{j+2+k}^{'} = d_{j+2+k}^{'} - d_{j+1+k}^{'} > 0 \quad (0\leq k<n-j-1) \tag{4}$$

在任务调度时，首先根据系统时钟的计时，更新差分时限链。在差分时限链中，只需将差分时限链首元素的时限减去计时时钟即可得到要插入任务的时限，这样就省去了为每个任务逐一进行时间更新的巨大时间开销。然后检测差分时限链首元素对应的任务是否临近其时限。若是，则将该任务优先级相应提高，而后进行基于优先级的抢占调度；否则，直接进行优先级调度。

在一次任务调度的开始，需通过系统时钟对差分时限链更新。当满足式(5)时，表明差分时限链首任务接近时限，应将其优先级相应提高。

$$0 \leq t_i^{'} - r_1 - \Delta \leq t_d \tag{5}$$

其中，$t_i^{'}$为更新后差分时限链链首元素的时限；r_1 为任务执行所需时间；Δ为每次优先级更新检测及任务上下文切换所需时间；t_d 为定义的判断系数，表示任务到达时限的接近程度。

差分时限优先任务调度算法的流程如图 1 所示。

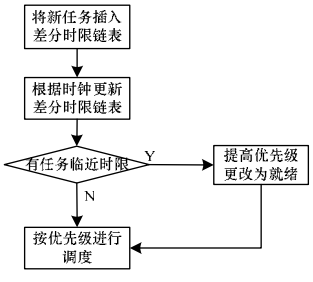


图 1　任务调度算法流程

当任务被抢占时，任务控制块中由执行时间变量(*ntime*)记录任务执行时间，并通过其值来更新其对应的任务在差分时限链中的时限值以备下一次检测。若任务在时限临近之前已被处理完，则在结束任务时将其时限块从差分时限链中断开。

1.2　同优先级任务间的调度优化

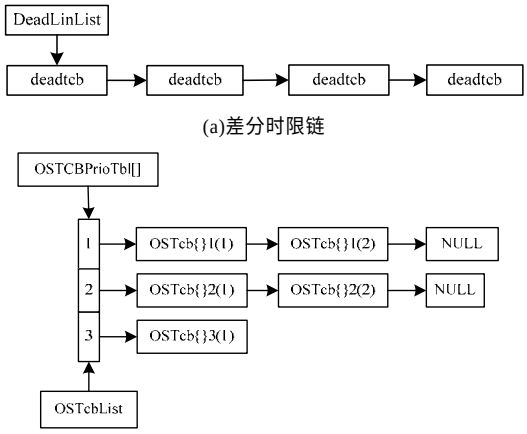
对 OSEK 操作系统中同一优先级的多个任务之间的协调处理可借鉴时间片轮转调度法。在基于优先级调度方式的基础上，为每一个优先级增加先进先出(First In First Out, FIFO)队列。在一个时间片结束后，进行优先级抢占式调度。若处于就绪态的最高优先级为当前任务的优先级，而且该优先级下有多个任务处于就绪态，则将该任务放入 FIFO 队尾，并为该优先级 FIFO 队首的任务分配一个时间片开始执行。否则，将该任务放回就绪表，取具有最高优先级任务的 FIFO 队首任务，开始执行。

2　改进算法在 uC/OS-II 内核的实现

2.1　任务管理机制的实现

本文在 uc/OS-II 操作系统中通过一个任务控制块的结构体——OS_TCB{}来表示一个任务，为了实现在一个优先级下有多个任务，在 OS_TCB{}中要添加一个指向同优先级任务的指针(*prioequ)，从而实现同优先级任务的 FIFO 调度策略。但要注意如果同优先级下有多个任务，任务就续表位图 OSRdyTbl[][4]相应的位必须在此优先级下的任务全部执行完后才能清零。

在不同优先级下加入了差分时限优先任务调度算法的机制，为了实现差分时限链，需为每个任务添加一个结构体(deadtcb)，其有任务 ID，前向指针(*Dpre)，后向指针(*Dnext)，任务控制块指针(*OSTCBDead)和时间限(deadline)4 个元素构成。在 OS_TCB{}中要添加一个指向差分时限链结构体的指针(*DeadTcb)，任务号(ID)2 个元素。改进后的任务结构如图 2 所示。



(a)差分时限链

(b)新的任务链接关系

图 2　任务管理结构

2.2　就绪算法的实现

根据进入就绪态任务的 prio，查找 OSTCBPrioTbl[]，如果该优先级下还没有任务，则通过 OSTCBList(指向任务链表的表头)把此任务加入到链表的表头，如果此优先级下已经有任务存在，则直接添加到 FIFO[5]队列的队尾，并根据任务的实现将任务添加到差分时限链中。

就绪算法的伪代码如下：

```
void OS_ReadyTask()
{ if(此优先级下 OSTCBPrioTbl[]没有置位)
    //此任务是该优先级下的第一个任务
      {通过 OSTCBList 将任务添加到任务链表表头；
       初始化任务控制块 TCB 中的相应项；
    将就绪表中的 OSRdyTbl[ ]和 OSTCBPrioTbl[]相应位置位；
    根据任务时限将任务添加到任务的差分时限链表中；}
else
   {找到同优先级任务链表的末尾，将任务添加到链表末尾；
    置新增任务的 next 指针为 NULL；
    根据任务时限将任务添加到任务的差分时限链中；}}
```

2.3　任务脱离就绪队列算法的实现

由于引入了同优先级的 FIFO 队列和任务差分时限链，

当任务脱离链表时，不能简单地对就绪表清零，对于不同优先级的任务，如果任务执行完成或挂起，则需要检查任务的差分时限链是否存在该任务，如果存在，需要在差分时限链中删除该任务；另外，需对于同优先级下是否存在多个任务加以判断；当满足条件：(1)在 prio 优先级下只有一个任务；(2)对于 OSRdyTbl[prio]所表示的 8 个优先级下均为空，才会将 OSRdyGrp[4]的某位清零，即表示在该位所表示的一组优先级下没有任务，而在同一优先级下有多个任务时，需要把指向同优先级链表头的指针后移一个节点，指向同优先级的下一个任务，以使当下一次调度产生的时候，调度器能以 $O(1)$ 的时间复杂度选取下一个任务，伪代码如下：

```
void OSClearReady( )
{根据 OSRdyGrp[ ]和 OSRdyTbl[ ]计算出最高优先级任务；
    if(此优先级下没有多个任务)
      {if(差分时限链表中的表头是此任务)
                {在差分时限链表中删除此任务；
                 把 OSRdyTbl[ ]相应位清零；
                   if(OSRdyTbl[ ]所表示的优先级下任务均没就绪)
                                {OSRdyTbl[ ]的相应位清零}}
        else {把 OSRdyTbl[ ]相应位清零；
                   if(OSRdyTbl[ ]所表示的优先级下任务均没就绪)
                                {OSRdyTbl[ ]的相应位清零}}
     else{同优先级链表的表头指向下一个任务；
          OSSched( )；}}
```

2.4 任务调度算法的实现

上文所述的差分优先任务调度算法，在 uC/OS-II 中需要做相应的改进。任务的时限，通过任务控制块中的执行时间变量(*ntime*)来计时，根据系统时钟来改变它；如果有任务达到时限，用系统的 OSTaskChangePrio()[3]来改变其优先级，并放入就绪队列，就续表的相应位置位。任务调度函数(void OSSched( ))的部分伪代码如下：

```
ntime++；
void OSSched( ){根据 ntime 更新差分时限链表；
        if(有任务到达时限)
          {用 OSTaskChangePrio()来改变其 prio；
          根据新 prio 将任务放入就绪队列；
          就续表相应位置位；
          根据任务 prio 进行任务调度；}
    else{直接根据任务的 prio 进行任务调度；}}
```

3 测试与结果分析

本文通过实例对差分时限优先任务调度算法与 EDF 算法、基于指定优先级的占先式任务调度算法作对比分析。在系统中构造 3 个任务，并根据其释放时刻建立任务，同时确定其优先级，执行时间和时限，用 D_i 来标识第 i 个任务的时限，如表 1 所示。

表 1 任务集 *S*

任务	指定优先级	释放建立时刻	执行时间/μs	时限/μs
*T*1	5	0	10	30
*T*2	8	4	10	25
*T*3	7	5	3	10

根据任务集 *S* 设计测试程序，分析 EDF 算法、基于优先级的抢占式任式务调度算法和差分时限优先任务调度算法。针对基于指定优先级的占先式任务调度算法，测试程序在任务切换时刻打印出原来运行的任务和执行时间；当总时间到达 20 μs 时，停止系统的运行。

在 EDF 算法中，对任务集 *S* 进行调度，任务 *T*1 运行到时刻 4，*T*2 释放，根据时限的接近程度分配优先级，因为 *D*2<*D*1，所以 *T*2 优先级高于 *T*1，*T*2 开始运行。在时刻 5，因为 *D*3<*D*2，所以 *T*3 优先级高于 *T*2，*T*3 开始运行，直至结束；*T*2 开始运行，最后 *T*1 运行。其中有 4 次任务切换，每次任务切换的时间包括查询就绪表和任务优先级更新时间、任务上下文切换时间，任务上下文切换时间对所有调度法是共有的，可省去不计。设查询就绪表和任务优先级更新时间为 Δt，可得出如表 2 测试结果中 EDF 栏的测试的结果。可以看出，*T*1 和 *T*2 都在时限内完成，而 *T*3 超时，没有得到执行。

表 2 测试结果

基于指定优先级的占先调度算法			EDF			差分时限优先任务调度算法		
时刻	已运行的任务	运行时间/μs	时刻	已运行的任务	运行时间/μs	时刻	已运行的任务	运行时间/μs
0	*T*1	0	0	*T*1	0	0	*T*1	0
4	*T*1	4	4	*T*1	4	4	*T*1	4
5	*T*2	1	5	*T*2	1-Δt	10	*T*2	6
14	*T*2	9	8	*T*3	3-2Δt	13	*T*3	3-Δt
20	*T*1	6	17	*T*2	9-3Δt	17	*T*2	4-Δt
…	…	…	23	*T*1	6-4Δt	23	*T*1	6-Δt

在差分时限优先任务调度算法中，对任务集 *S* 进行调度，任务 *T*1 运行到时刻 4，*T*2 释放，由于 *T*2 指定的优先级高于 T1，且此时没有任务临近时限，则 *T*2 抢占 *T*1 开始执行。在时刻 5，*T*3 被释放，由于 *T*3 的指定优先级低于 T2 且没有任务临近时限，*T*2 继续执行。当到达时刻 10，任务 *T*3 临近时限，将 *T*3 的优先级相应提高(设提高到 9)，高于 *T*2 的优先级，因此 *T*3 开始执行。在时刻 15，*T*3 结束运行，*T*2 高于 *T*1 的优先级，开始运行。到时刻 19，*T*2 运行结束，*T*1 继续运行，直至结束。整个过程只进行了一次查询就绪表和任务优先级更新如表 2 测试结果中差分时限优先任务调度算法栏的结果。

从测试结果中可以看出，基于指定优先级的任务调度算法不能保证低优先级的任务在时限内得以执行，从而大大提高了系统的性能和实时性。

4 结束语

本文改进的任务调度机制在取得对同优先级多任务调度支持和保证任务在时限前能获得执行的同时，牺牲了部分任务响应效率，增加了部分延时。在多次测试中，最差的任务响应时间有较大差别，但基本能满足汽车电子对实时性需求。

参考文献

[1] OSEK/VDX. Operation System Specification 2.2.3[Z]. (2005-02-17). http://www.OSEK-VDX.org.

[2] 杨立身，王中海. 嵌入式实时操作系统任务调度算法改进[J]. 微型电脑应用, 2007, 23(9): 44-46.

[3] 谢　敏，李桥梁. 嵌入式实时操作系统任务调度算法优化[J]. 电子科技, 2005, (12): 24-26.

[4] 陈　卓，熊忠阳. 基于 OSEK/VDX 操作系统的任务管理机制设计[J]. 计算机工程, 2006, 32(12): 82-84.

[5] 苏　娟，吴旭光，张　朝. 嵌入式操作系统 μC/OS-Ⅱ内核改进及其应用扩展[J]. 计算机工程, 2007, 33(15): 61-63.

编辑　金胡考