时间车轮算法在 μC/OS－Ⅱ中的应用

Application of Time Wheels Algorithm on μC/OS－Ⅱ

刘 晋 孙 楠 （辽宁师范大学 计算机与信息技术学院 辽宁 大连 116029）

摘 要： 通过分析嵌入式操作系统 μC/OS－Ⅱ中时钟节拍处理的不足，本文提出一种应用时间车轮算法对其进行改进的思想。时间车轮算法有效地提高了时钟中断响应速度和 μC/OS－Ⅱ的时钟精确度。

关键词： 时钟节拍 时间车轮 延时 μC/OS－Ⅱ

1 引言

μC/OS－Ⅱ是一种开放源码的实时嵌入式操作系统内核（RTOS），具有占先式、多任务的特点，已被应用到众多的微处理器上，用于建立和提供比单任务应用更好的响应系统，通常作为处理复杂性需求的首选。μC/OS－Ⅱ的调度器是基于优先级的，即 CPU 的控制总是给予就绪态的最高优先级的任务。因此，在 μC/OS－Ⅱ对任务进行调度和管理的过程中，任务延迟和唤醒时间的性能好坏就显得尤为重要。任务由等待变为挂起直到某一时间结束，需要经过一个可变的时间，由时钟节拍（Clock－Tick）计数。在对 μC/OS－Ⅱ的分析和使用中发现，其内核在任务调度和管理中，由于时钟节拍处理的缺陷，导致进程延迟与唤醒的处理方面尚有不合理之处，本文运用时间车轮算法对其进行相应的改进。

2 μC/OS－Ⅱ中时钟节拍的处理

μC/OS－Ⅱ可以管理 64 个任务。任务一旦被建立，系统将从空任务链表中分配一个任务控制块 OS_TCB 给该任务。建立的所有任务包括系统自动建立的都放在双向的任务链表 OSTCBList 中。任务的状态分为睡眠态、就绪态、运行态、等待态和中断服务态。正在运行的任务可以通过时间延时函数 OSTimedly（），将自身延迟一段时间，这个任务将进入等待状态，一直到函数中定义的延时时间到。延迟时间以节拍数为单位。“计时”工作由时钟节拍函数 OSTimtick（）完成。

在时间延迟函数 OSTimedly（）中，将延时的时钟节拍数赋值给任务控制块中的延时项 OSTCBDly。时钟节拍函数 OSTimtick（）中大量的工作是给每个用户任务控制块 OS_TCB 中的时间延时项减 1（直到该项延时为零）。OSTimetick（）从 OSTCBList 开始，沿着 OS_TCB 链表扫描任务进行逐个减 1，一直做到空闲任务。当某任务的任务控制块中的时间延时项 OSTCBDly 减到了零，则证明这个任务延时时间到，任务就进入就绪态。

在沿 OSTCBList 扫描时，需要不断地对每个任务控制块的时间延时项减 1，建立了多少任务就要对多少个任务进行处理。这种方法的优点是扫描整个 OSTCBList 的执行时间是确定的，但是这其中对已经就绪的任务和正在执行的任务的处理是没有必要的。这增加了系统延时时间，降低了系统的效率。这样一来，由于程序中的延时和等待，浪费了大量的 CPU 时间，严重影响了 CPU 能力的发挥。

μC/OS－Ⅱ时钟节拍处理的另一个不足在于，时钟节拍函数 OSTimtick（）对任务链表进行扫描的过程必须在关中断的情况下完成的，如果扫描时有中断访问将严重影响时钟节拍处理的结果，甚至引起错误处理。μC/OS－Ⅱ作为一种实时的操作系统，这种受限的处理无疑是致命的缺陷。

本文针对 μC/OS－Ⅱ以上的两点不足，运用时间车轮算法，将其应用于对该操作系统时钟节拍的处理上，对其进行了有效的改进。

3 应用时间车轮算法的改进

时间车轮是一个固定长度的数组，每个结点代表一个具有精确软件计数器设置的时间单位。时间车轮方法具有分类时钟列表的优点，并且更新计时器效率高。

© 中国科学院软件研究所 http://www.c-s-a.org.cn

应用在 μC/OS－II 中具体的方法为：首先，时钟节拍数为 16 位二进制数（即包含 4 个四位的数），可以建立 4 个容量为 16 的数组来表示给定的节拍数，并约定初始时 4 个数组指针分别指向 4 个数组的首地址（如图 1 所示）。然后，以 4 个数组为车轮按照 CPU 时钟节拍进行轮转，修改 μC/OS－II 的延时函数和时钟节拍函数，达到在无需关中断的情况下多任务的延时唤醒处理。

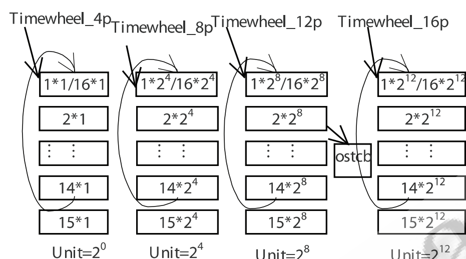

图 1　时间车轮算法示例

3.1 延时函数 OSTimeDly( ) 的修改

由于已经设定延时节拍数为 16 位整数，且从低位到高位每 4 位为一组分为容量为 16 的 4 组，因此可以按组分别进行延时处理。例如有延时时间 858 个时钟节拍（即 0000001101011010），由低位到高位分组为 1010、0101、0011、0000（即 10、5、3、0），此时，延时函数 OSTimeDly( ) 就可以被修改为包含四个独立延时操作的集合。

任务被延时时，四个延时操作依次启动，如果该组的延时节拍数为 0，则不需在该组链接任务；否则，将任务链接到该组对应的位置上，然后任务就等待延时期满进入就绪态。如第一组中延时节拍数为 10，则在任务建立的时钟位置链接该任务，并延时 10 个时钟节拍后进入就绪态；而第四组中延时节拍数为 0，则不需链接该任务。

当然，在多任务的情况下，会出现多个任务指向同一数组同一位置的现象，本方法中将这些任务按顺序依次链接，形成具有相同位置指针的单链表，在接下来的时钟节拍计数中只需按照单链表的链接顺序进行扫描挂起即可。

3.2 时钟节拍函数 OSTimeTick( ) 的修改

本方法中，时钟节拍函数通过使用指针扫描数组的方式来标记时钟节拍数。由于低位向高位有进位的关系，特别的，在本文设定的时间车轮算法中，当低位车轮轮满 16 个时钟节拍便使邻近高位车轮的轮转自动增加一位。例如，第一个时间车轮数组的指针从第 0 位开始，每过一个时钟节拍指针向下移动一位，当指针轮到第 15 位后将重新指向第 0 位，此时第二个数组的指针便增加一位，即向下移动一位。依此类推，第二个数组轮满时第三个数组的指针向下走一位。同理可得第三个数组向第四个数组的“进位”。

时钟节拍数标记的同时，每个数组中车轮的轮转是不受影响的，即数组的指针在逐位地扫描本组所链接的任务。每当指针到达一个位置，都要检查其后是否链接有等待延时的任务，如果有就使其挂起进入就绪态。

这样就可以在无需关中断的情况下完成任务的延时和唤醒，并避免了由于逐项扫描减 1 而造成的 CPU 时间浪费。

4　总结

本算法的优点在于应用时间车轮函数改进原有的时间节拍函数，每个时钟节拍服务只是修改时间车轮的指针，如果槽内有等待任务则将它们就绪，无需进行耗费时间的逐项扫描递减，大大节省了 CPU 在空闲等待的时间，提高了 CPU 的工作效率；其次，时间车轮函数的执行无需系统关中断，允许处理过程中有中断访问，并能无阻碍的处理，从而提高了 CPU 的利用率。

但是，由于本方法设定的四个车轮只能分别表示 16 个时钟节拍的容量，当时间延迟溢出时只能重新开始计数而不是溢出的响应位置，即本方法在溢出情况的处理上尚有误差，因此，提高时间精度将成为进一步的工作。

参考文献

1 Jean J. Labrosse. μC/OS－II—源码公开的实时嵌入式操作系统. 邵贝贝译. 北京：中国电力出版社，2001. 90－93. 127－132.

2 Li Q, Yao C. 嵌入式系统的实时概念. 王安生译. 北京：北京航空航天大学出版社，2004. 149－165.

© 中国科学院软件研究所　http://www.c-s-a.org.cn